UNIVERSIDADE FEDERAL DE SANTA CATARINA

Desenvolvimento de aplicações sob o paradigma da computação em nuvem com ferramentas Google

Victor Daniel Müller

Florianópolis - SC

2010

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Desenvolvimento de aplicações sob o paradigma da computação em nuvem com ferramentas Google

Victor Daniel Müller

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Florianópolis - SC

2010

Victor Daniel Müller

Desenvolvimento de aplicações sob o paradigma da computação em nuvem com ferramentas Google

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Ricardo Felipe Custódio

Co Orientador: Roberto Samarone dos Santos Araújo

Banca examinadora

_____

Marcelo Brocardo

_____

Julíbio David Ardigo

_____

Júlio da Silva Dias

Agradecimentos

Agradeço aos meus pais por todo o esforço que fizeram para possibilitar que eu pudesse empreender este projeto. Agradeço também à minha namorada e à sua família pela compreensão e ajuda que recebi durante o desenvolvimento do trabalho.

Um agradecimento especial ao professor Ricardo Felipe Custódio pela orientação que obtive para desenvolver o trabalho.

Resumo

O crescente avanço no desenvolvimento das tecnologias web possibilitou uma nova abordagem sobre o desenvolvimento web, nomeada de computação em nuvem, despertando para uma nova possibilidade de desenvolvimento e distribuição de aplicações. Baseado nesta abordagem é feita uma análise das ferramentas disponibilizadas pelo Google através do Google App Engine e Java para desenvolvimento de aplicações escaláveis, além de ferramentas que sirvam de auxílio para aplicações web como estas.

Palavras-chave: Computação em nuvem, SaaS, PaaS, IaaS, App Engine, GFS, BigTable, Google Web Toolkit, Java, JDO.

Abstract

The increasing progress in the development of web technology has enabled a new approach to web development, named cloud computing, awakening to a new possibility of development and distribution of applications. Based on this approach, an analysis is made of the tools provided by Google through the Google App Engine and Java for development of scalable applications, and tools that serve as aid for web applications like these.

Palavras-chave: Cloud Computing, SaaS, PaaS, IaaS, App Engine, GFS, BigTable, Google Web Toolkit, Java, JDO.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

AJAX	Asynchronous Javascript And XML
API	Application Programming Interface
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DTO	Data Transfer Objects
EC2	Amazon Elastic Compute Cloud
FIFO	First In, First Out
GFS	Google File System
GHZ	Gigahertz
GMT	Greenwich Mean Time
GQL	Google Query Language
GWT	Google Web Toolkit
HTM HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol secure
IAAS	Infrastructure as a Service
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
JAVA SE	Java Standard Edition
JDK	Java Development Kit
JDO	Java Data Object
JPA	Java Persistence API
JSP	Java Server Pages
JVM	Java Virtual Machine
MVC	Model View Controller
PAAS	Platform as a Service
PHP	Hypertext Preprocessor
POJO	Plain Old Java Object
RPC	Remote Procedure Call
SAAS	Software as a Service
SDK	Software Development Kit
SQL	Structured Query Language
SVN	SubVersioN
URL	Uniform Resource Locator
WAR	Web Application Archive
XMPP	Extensible Messaging and Presence Protocol

1. Introdução

Com a evolução de diversas tecnologias computacionais correlatas, como a computação distribuída, internet e linguagens de programação, tornou-se possível o surgimento e exploração de novas áreas da computação.

A idéia de vender recursos computacionais sob demanda, de acordo com a necessidade do cliente é uma idéia antiga, porém se tornava limitada às grandes instituições capazes de adquirir recursos computacionais das grandes detentoras de poder computacional.

Com o passar do tempo, e conseqüente desenvolvimento de novas tecnologias nos mais abrangentes campos da computação, principalmente as relacionadas à massificação do uso da internet a idéia voltou à tona. Primeiramente procurou-se disponibilizar aplicativos web que poderiam ser acessados de qualquer lugar através da internet. Mais adiante, o desenvolvimento de tecnologias de virtualização viabilizou a venda sob demanda e de forma escalável de infraestrutura e recursos computacionais capazes de sustentar estes aplicativos web.

O agrupamento destas áreas sob uma nova ótica fez surgir o paradigma da computação em nuvem, gerando a tendência cada vez maior de aplicativos que possam ser acessados de forma eficiente de qualquer lugar. Este paradigma criou a necessidade de repensar o modo como as aplicações são desenvolvidas e disponibilizadas, ao mesmo tempo em que motivou o desenvolvimento de tecnologias capazes de dar suporte ao seu aprimoramento.

O paradigma da computação em nuvem vem ganhando cada vez mais força com a adesão de grandes empresas do ramo da computação, que vem realizando cada vez mais esforços no desenvolvimento de tecnologias a ele relacionadas.

1.1 Objetivos

1.1.1 Objetivo geral

Este trabalho tem como objetivo geral realizar um estudo sobre o paradigma da computação em nuvem e como isto pode ser posto em prática com o uso e análise de ferramentas desenvolvidas pelo Google.

1.1.2 Objetivo específico

Os objetivos específicos deste trabalho visam:

a) Conceituar o paradigma da computação em nuvem;
b) Análise das ferramentas disponibilizadas pelo Google para desenvolver aplicações sob este paradigma;
c) Análise de ferramentas de desenvolvimento auxiliares;
d) Projetar e implementar uma aplicação a fim de experimentar a ferramenta;

1.2 Justificativa

O crescente interesse no desenvolvimento de aplicações web tem mostrado o paradigma da computação em nuvem como a tendência para o futuro da computação de um modo geral.

Por se tratar de um tema relativamente novo e de grande potencial, há uma demanda de trabalhos relacionados e que possam contribuir para o seu desenvolvimento.

1.3 Motivação

A tendência geral para o desenvolvimento web como o futuro das aplicações, bem como os esforços de grandes empresas de tecnologia neste sentido abrem enormes caminhos para a exploração desta área ainda carente de publicações.

1.4 Estrutura do trabalho

Este trabalho está organizado em 8 capítulos bem definidos. O capítulo 2 apresenta aspectos relacionados ao paradigma da computação em nuvem. O capítulo 3 apresenta uma análise da plataforma Google Application Engine, desenvolvida pelo Google para desenvolver aplicativos que utilizem sua infraestrutura. O capítulo 4 aborda o uso desta plataforma com ferramentas para desenvolvimento utilizando a linguagem Java. O capítulo 5 mostra como a ferramenta Google Web Toolkit pode auxiliar no desenvolvimento de aplicações mais completas de modo integrado ao Google Application Engine. O capítulo 6 mostra o projeto e desenvolvimento de uma aplicação de exemplo que utiliza estas ferramentas. O capítulo 7 traz uma discussão sobre o que foi apresentado no trabalho. Por fim o capítulo 8 encerra o trabalho com algumas considerações finais.

2. Computação em Nuvem

2.1 Introdução

Este capítulo introdutório traz um breve histórico e atual estado dos sistemas baseados em computação em nuvem, bem como abre portas para a mudança na abordagem de disponibilização e comercialização de aplicativos e recursos computacionais.

2.2 Definição

O uso do termo C*loud Computing* ou *Computação em Nuvem* como também é conhecido no Brasil, vem se tornando cada vez mais freqüente, com a promessa de ser um paradigma que irá mudar a forma como os softwares são construídos e comercializados. Porém tão nebuloso quanto seu nome é a sua definição, sendo ainda causa de muitas divergências quanto à utilização deste termo, muito devido ao fato de ser um conceito muito recente.

Apesar de não haver um consenso sobre a exata definição do que é *Computação em Nuvem*, FOSTER possui uma definição interessante e bastante abrangente para este paradigma, podendo ser tomada como referência:

> *"Computação em nuvem é um paradigma de computação em larga escala que possui foco em proporcionar economia de escala, em que um conjunto abstrato, virtualizado, dinamicamente escalável de poder de processamento, armazenamento, plataformas e serviços são disponibilizados sob demanda para clientes externos através da internet."*

Tomando esta definição como referência, serão abordados os principais assuntos acerca de *computação em nuvem,* bem como sua relação com tecnologias já existentes.

2.3 Tipos de serviço

Um conceito importante inerente a computação em nuvem se refere ao modo como os diversos serviços disponíveis através da sua infraestrutura são disponibilizados ao usuário, seja ele desenvolvedor ou usuário final.

A idéia básica ao redor do computação em nuvem consiste em que tudo é serviço, uma espécie de *self-service* de tecnologias da informação. Muitos serviços de computação em nuvem *s*ão oferecidos pelo modelo conhecido como *Utility Computing*, o qual é definido como um pacote de recursos computacionais medidos e cobrados de forma semelhante aos serviços de utilidade pública, como eletricidade, água ou telefone. Este modo prevê um melhor investimento nos recursos de hardware, diminuindo a necessidade de grandes investimentos iniciais, possibilitando o aluguel de mais recursos à medida que forem necessários maiores recursos computacionais, podendo ainda alguns provedores de serviços se ajustarem dinamicamente às necessidades de demanda, ou seja, a possibilidade do cliente contratar recursos extras apenas para momentos de pico, evitando que recursos fiquem ociosos a maior parte do tempo.

Assim como a própria computação em nuvem, o *Utility Computing* não é um conceito necessariamente novo, mas que vem ganhando visibilidade atualmente. John McCarthy, criador da linguagem *LISP*, já vislumbrava em 1961 o dia em que os recursos computacionais seriam disponibilizados da forma como são feitos os serviços de utilidade pública, podendo assim ocupar um novo nicho de mercado. Durante as décadas seguintes empresas como a *IBM* e outras fornecedoras de mainframes seguiram esta linha de serviços, basicamente fornecendo poder computacional e recursos de armazenamento para bancos e outras grandes instituições. No final da década de 90 o *Utility Computing* ressurgiu*,* principalmente devido a grande força que a computação em nuvem tem ganhado desde então como paradigma emergente, já que os dois termos encontram-se estritamente ligados.

Atualmente os serviços de *computação em nuvem* possuem três categorias bem definidas de acordo com os recursos e o modo que estes recursos são disponibilizados:

a) Infraestrutura como um Serviço (IaaS)
b) Plataforma como um Serviço (PaaS)
c) Software como um Serviço (SaaS)

Olhando atentamente a cada uma delas nos damos conta que muito provavelmente já utilizamos algum serviço conceituado nesta tecnologia, sem nos darmos conta de que se tratava de computação em nuvem. Isto se dá principalmente pelo fato de que não se trata efetivamente de uma nova tecnologia, mas sim de um conjunto de tecnologias já existentes aperfeiçoadas, agrupadas sob a ótica de um novo paradigma e com todo um marketing por trás.

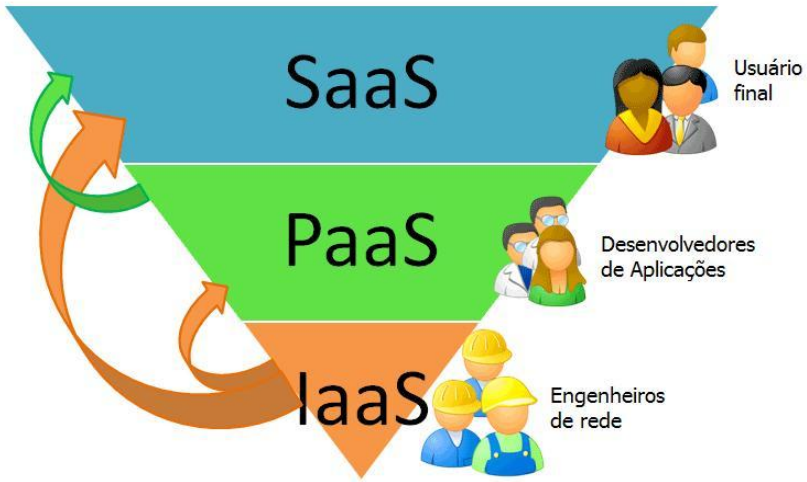

Figura 2.1: Estrutura de serviços de computação em nuvem. (Traduzido de: Cloud computing, a nuvem que se aproxima 2008)

2.3.1 Infraestrutura como um Serviço

A Infraestrutura como um serviço (IaaS), é a base dos serviços de computação em nuvem sendo a parte que se refere à disponibilização dos recursos de hardware, formados pelos servidores, armazenamento e processadores.

Originalmente este ramo de serviços foi denominado *Hardware as a Service (HaaS)* pelo economista Nicholas Carr (2006), mas tal termo foi substituído com a gradual adoção desta idéia por parte de empresas provedoras de serviços web, culminando em seu aprimoramento e fornecimento de fato de uma total infraestrutura integrada capaz de suprir demandas de armazenamento de informação, poder de processamento, largura de banda, etc.

Geralmente o fornecimento destes serviços se apóia em tecnologias como a de virtualização, onde o consumidor do serviço adquire uma máquina virtual operando sobre a estrutura de servidores do provedor, sendo a quantidade de processamento e armazenamento contratados disponibilizados através da configuração adequada de tais máquinas. Este modelo diminui drasticamente os obstáculos causados por uma possível demanda de mais recursos computacionais, bem como os custos que isto implica, já que para isto basta configurar as máquinas virtuais, sem a necessidade de uma possível migração de dados para novos servidores, ou gastos com a aquisição dos mesmos. O ponto chave destes serviços se encontra na facilidade de contornar problemas de escalabilidade dos sistemas, ou seja, na grande capacidade de suportar demandas crescentes por recursos de forma uniforme, bastando para isto apenas ajustar máquinas virtuais, característica bastante conhecida no meio da computação em nuvem como “elasticidade”.

Atualmente os serviços de IaaS vêm se tornando uma tendência crescente dentre as empresas de hospedagem de sistemas web, sendo possível notar uma oferta cada vez maior de produtos com as características deste ramo de computação em nuvem devido principalmente ao grande número de benefícios que esta abordagem pode trazer.

2.3.2 Plataforma como um Serviço

Plataforma como um Serviço (PaaS) é um tipo de serviço que procura prover toda a estrutura necessária para um ambiente de desenvolvimento. O alvo destes serviços são os desenvolvedores de software, tornando disponível uma plataforma computacional completa, agregando sistema operacional, linguagens de programação, bibliotecas, sistemas gerenciadores de banco de dados, etc. Neste serviço o ambiente de desenvolvimento em si está disponível através da nuvem, deste modo os desenvolvedores podem construir suas aplicações sem a necessidade de instalar qualquer ferramenta no seu computador e de distribuí-la de forma simplificada. Uma alternativa ao PaaS seria desenvolver as aplicações usando ferramentas de desenvolvimento para *desktops* e depois enviá-las para um provedor baseado em computação em nuvem .

As plataformas PaaS possuem algumas diferenças funcionais em relação às plataformas de desenvolvimento tradicionais. Algumas delas são:

a) Ferramentas de desenvolvimento multiusuário: ferramentas de desenvolvimento tradicionais são monousuário, um ambiente baseado em PaaS deve suportar vários usuários, cada um com múltiplos projetos ativos;
b) Arquitetura de distribuição multiusuário: geralmente a escalabilidade não está incluída no escopo de desenvolvimento do projeto, deixando este assunto a cargo dos arquitetos, enquanto isto é algo inerente ao PaaS;
c) Gerenciamento integrado: soluções tradicionais geralmente não chegam a um consenso sobre monitoramento em tempo real, mas no PaaS a habilidade de monitorar deve fazer parte da plataforma de desenvolvimento;
d) Cobrança integrada: PaaS requer mecanismos específicos de cobrança baseados no uso.

Para se caracterizar um bom serviço de PaaS alguns requisitos são imprescindíveis. Deve-se ter disponível um ambiente completo e totalmente integrado de desenvolvimento capaz de fornecer suporte ao desenvolvimento, teste, distribuição, hospedagem, manutenção e monitoramento do software, otimizando assim o ciclo de vida do software. Em um serviço completo de PaaS todo o ciclo de vida do software deve ser suportado no mesmo ambiente computacional, reduzindo drasticamente os custos de desenvolvimento e manutenção, o tempo até possuir um aplicativo pronto para ser distribuído e os riscos de projeto. Este alto nível de integração proporciona que uma maior parte de energia criativa seja destinada à concepção da aplicação ao invés de ocupar grande parte do tempo com configuração ou integração das mais diversas ferramentas que constituem os ambientes padrões, ao passo de que todo o processo se dá de forma simplificada. Portando os ambientes PaaS chegam para buscar facilitar a vida de desenvolvedores de software acelerando as iterações de desenvolvimento e tornando o desenvolvimento das aplicações mais rápido e intuitivo.

2.3.3 Software como um Serviço

Certamente Software como um serviço é o tipo de serviço de computação em nuvem com o qual a maioria das pessoas já se deparou muitas vezes sem se dar conta de que se trata de um serviço baseado neste paradigma. Utilizando uma visão mais abrangente dos conceitos de SaaS podemos citar uma variedade de exemplos de aplicações em que neles se enquadram, como é o caso das aplicações de webmail – aplicações que tornaram praticamente dispensáveis os aplicativos clientes de e-mail desktop convencionais –, sites de compartilhamento de fotos e vídeos, aplicações de escritório (processadores de texto e planilhas) como as disponibilizadas pelo Google através do *GoogleDocs, e* aplicações de âmbito corporativo como é o caso das aplicações CRM, as quais servem para gerenciar o relacionamento entre a empresa e seus clientes, fornecidas pelo Salesforce.com e Microsoft, ou seja, todas as aplicações disponibilizadas através da web. Essa inovação no modo como as aplicações são distribuídas foi conquistada devido ao

grande avanço pelos quais as tecnologias web sofreram ultimamente, principalmente com o advento da chamada *Web 2.0*.

Uma definição simplista para o modelo Software como um Serviço poderia ser a de software distribuído como um serviço hospedado e acessado através da internet. Porém se levarmos em conta apenas esta característica não teremos a especificação de nenhuma arquitetura – não faz referência a nenhuma tecnologia ou protocolos específicos – ou modelo de negócios específico, apenas define onde o código da aplicação reside, e como eles são distribuídos e acessados. Portanto, seguindo esta definição basta que a aplicação atenda ao seguinte critério: um fornecedor hospeda toda a lógica do programa e os dados, e provê acesso aos usuários finais através da internet por uma interface web.

Se olhado mais atentamente, pode-se identificar duas categorias principais de Software como um Serviço*:*

a) Serviços *line-of-bussines:* oferecidos à empresas e organizações de todos os tamanhos. Geralmente estes tipos de serviços são grandes, soluções customizadas focando as facilidades dos processos de negócio como as finanças, gerenciamento e relações com clientes (CRM). Estes serviços são geralmente vendidos aos clientes sob uma assinatura;
b) Serviços orientados ao consumidor: são oferecidos ao público comum. Tais serviços, assim como o anterior, são fornecidos sob algum tipo de assinatura, mas geralmente são fornecidos sem custo, sendo suportados por anúncios de publicidade.

Portanto para mudar do modelo tradicional de distribuição de software para o modelo de Software como um serviço requer dos vendedores mudanças no modo de pensar em três áreas: modelo de negócios, arquitetura da aplicação e estrutura operacional.

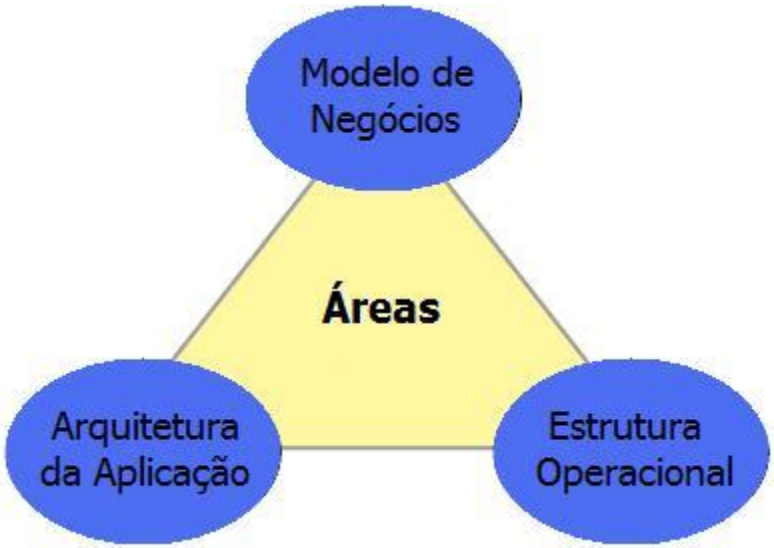

Figura 2.2: Áreas que diferenciam o modelo SaaS. (Traduzido de: CHONG e CARRARO 2006)

2.3.3.1 Modelo de negócios

Mudar o modelo de negócios poderia significar a mudança da propriedade do software do cliente para um provedor externo, a responsabilidade pelo gerenciamento da infraestrutura tecnológica – isto é, hardware e serviços de profissionais especializados – do cliente para o provedor, reduzir o custo de disponibilização de serviços de software, através da especialização e da economia de escala.

Muitos softwares continuam a ser vendidos do mesmo modo por décadas. O cliente compra uma licença para o uso do software, instala no seu hardware, com o vendedor fornecendo suporte de acordo com os termos da licença. Com o modelo SaaS a idéia é a de que, ao invés de possuir o software diretamente, o cliente paga por uma assinatura do software rodando nos servidores de alguém, perdendo o direito de usufruí-lo quando interromper a assinatura.

Do ponto de vista da responsabilidade acerca da infraestrutura tecnológica, podemos dizer que, na visão tradicional, a maior parte do orçamento referente a este assunto em uma organização se resume a três grandes áreas:

a) Software: programas e dados que as organizações utilizam para processamento da informação;
b) Hardware: desktops, servidores, componentes de rede e dispositivos móveis que provém aos usuários o acesso ao software;
c) Profissionais: pessoas e instituições que garantem a operação e disponibilidade do sistema, incluindo suporte técnico, consultores e representantes de venda.

Destas três áreas, as duas últimas são destinadas a dar suporte para a primeira, fazendo o possível para que o software produza o resultado final desejado e eficiente. Deste modo, é correto afirmar que a maior parte do orçamento é gasto em hardware e serviços de profissionais, deixando a menor parte do orçamento para gastos com software. Os gastos com software se resumem a licenças e customização, o orçamento gasto com hardware é destinado à aquisição de desktops ou laptops para usuários finais, servidores para hospedagem de dados e aplicações e componentes de rede, por fim, os gastos com serviços profissionais são com suporte para distribuição, software e hardware, assim como com consultoria e desenvolvimento de recursos para sistemas customizados.

Figura 2.3: Orçamento destinado às principais áreas do ambiente de TI. (Traduzido de: CHONG e CARRARO 2006)

Em uma organização que se apóia principalmente sobre o modelo SaaS, a alocação de orçamento se torna bem diferente. Deste modo o fornecedor do software SaaS armazena aplicações e dados associados nos seus servidores,

tirando do cliente a responsabilidade por aquisição, suporte e manutenção do software hospedado e do hardware requerido por ele. Além disso, este modelo demanda por menos investimentos em desktops do que as aplicações locais tradicionais, otimizando o ciclo de vida do hardware envolvido significativamente, significando em um aumento do orçamento destinado a gastos com software, geralmente utilizados com taxas de assinatura de serviços SaaS.

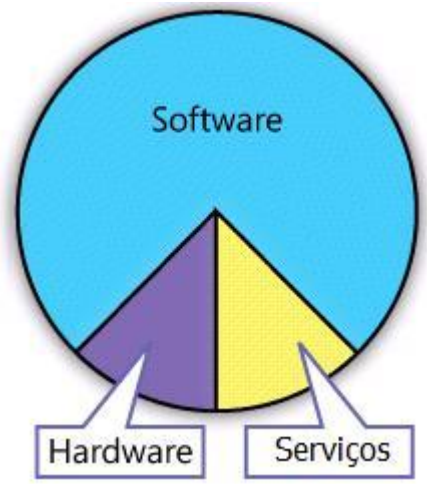

Figura 2.4: Orçamento utilizando o modelo SaaS. (Traduzido de: CHONG e CARRARO 2006)

Contudo, este modelo de negócios gerar a dúvida destes resultados serem uma ilusão, afinal uma porcentagem das taxas de assinatura pagas aos fornecedores de serviços SaaS tem que pagar pelo hardware e serviços de profissionais destes fornecedores. A resposta para esta questão está na economia de escala. Tomemos como exemplo um fornecedor que possui uma aplicação instalada e distribuída sobre cinco servidores capazes de suportar 50 clientes, isto significa que cada cliente é responsável por um décimo do custo de um servidor.

Uma aplicação similar instalada localmente iria requerer de cada cliente dedicar um servidor inteiro para a aplicação, ou até mais, a fim de aumentar o grau de balanceamento e disponibilidade. Deste modo o modelo SaaS representa uma economia substancial, com o custo operacional por cliente tendendo a diminuir à medida que mais clientes são incorporados. Com a gradual adoção deste modelo a tendência é que tenhamos serviços de maior qualidade a preços cada vez menores. Portanto, mesmo com os custos de hardware e gastos com profissionais embutidos

no preço final dos serviços é possível uma redução nos gastos em relação ao modelo tradicional ou então a obtenção de softwares mais completos utilizando-se do mesmo orçamento.

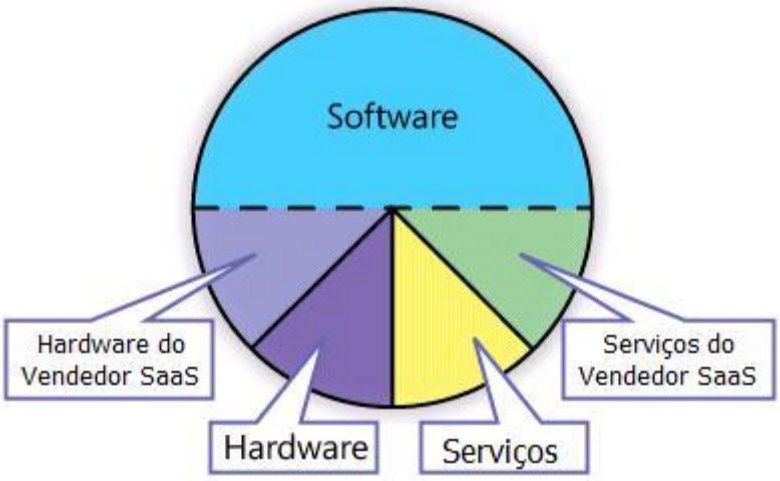

Figura 2.5: Orçamento utilizando o modelo SaaS apontando os gastos embutidos. (Traduzido de: CHONG e CARRARO 2006)

Deste modo, softwares fornecidos sob o modelo SaaS podem atingir clientes que antes não eram alcançados em certos nichos de mercado. Um caso prático são os aplicativos corporativos, geralmente feitos sob medida e podendo necessitar de assistência para a instalação e visitas de suporte, além de hardware dedicado. Toda esta atenção necessária ao funcionamento do software contribui para estabelecer o preço mínimo que o vendedor pode vender o software, restringindo os possíveis clientes às grandes empresas capazes de arcar com tais custos.

Por outro lado, para cada grande empresa existe um grande número de pequenos e médios negócios que poderiam se beneficiar de uma solução, mas não podem arcar com os custos. Assim eliminando grande parte dos custos com manutenção através da economia de escala, vendedores SaaS podem oferecer soluções por um custo muito menor que as tradicionais, possibilitando que novos clientes em potencial estejam ao alcance e que antes estavam inacessíveis por conta dos custos.

Efetivamente focar estes clientes menores requer mudanças no ponto de vista do processo de venda, automatizando de forma que não sejam necessárias relações interpessoais para isso, possibilitando que o cliente acesse o website do serviço, pague com um cartão de crédito, configure o serviço de modo a atender suas necessidades e por fim possa usá-lo, sem intervenção humana por parte do vendedor para isso. Esta abordagem, no entanto não exclui a existência de relações mais pessoais com clientes maiores e com necessidades mais amplas, mas com a diferença de despender um esforço menor para tarefas deste gênero do que no modelo tradicional.

2.3.3.2 Arquitetura da aplicação

A definição usada até então para Software como um Serviço é a de software distribuído como um serviço hospedado e acessado através da internet. No entanto esta definição um tanto quanto abrangente não é muito útil para um arquiteto de software, pois não esclarece o que realmente faz uma boa aplicação SaaS funcionar. Precisamos então definir o que podemos chamar de uma aplicação SaaS madura, introduzindo alguns critérios adicionais.

Do ponto de vista da arquitetura da aplicação, podemos mencionar três atributos essenciais que diferenciam uma aplicação SaaS bem projetada de uma pobre neste aspecto. Portanto uma boa aplicação SaaS deve ser: escalável, eficiente do ponto de vista de atender multiusuários e configurável.

Para a aplicação ser escalável, ela deve procurar maximizar a concorrência e utilizar os recursos da forma mais eficiente possível. Isto pode representar otimização no uso de bancos de dados, armazenamento de dados em cache ou compartilhamento de recursos, como threads e conexões de rede.

2.3.3.3 Estrutura operacional

Do ponto de vista da estrutura operacional, quando se fornece uma aplicação como serviço que está hospedada em uma infraestrutura que cobra pelo uso

individual dos recursos é necessário repassar este custo ao consumidor de uma maneira eficiente, a qual pode ser por meio de um sistema de cobrança baseado no uso.

Outro ponto importante a considerar em relação à estrutura operacional diz respeito à necessidade de mecanismos capazes de monitorar e garantir a disponibilidade e desempenho do aplicativo, de modo que a aplicação seja vantajosa e atrativa aos clientes.

2.4 Conclusão

O crescente desenvolvimento de tecnologias voltadas à internet suscitou uma nova e abrangente área da computação, que vem cada vez mais se estabelecendo como a tendência para o futuro para o desenvolvimento de aplicações.

Este novo paradigma conhecido como computação em nuvem despertou a necessidade para o desenvolvimento de novas tecnologias e abordagens no desenvolvimento e distribuição de aplicações, bem como tecnologias de suporte e infraestrutura revelando-se um campo com grande potencial de desenvolvimento.

3. Google Application Engine

3.1 Introdução

O Google Application Engine, ou App Engine como é comumente chamado é um conjunto de ferramentas e serviços disponibilizado pela empresa norte americana Google.

Trata-se de um modelo de PaaS, que diferentemente de seu conceito original, em que todo o ambiente responsável pelas etapas de desenvolvimento e publicação do software se dá através de ferramentas disponibilizadas via web, disponibiliza um ambiente desktop completo e de fácil configuração para esta finalidade.

Através delas o desenvolvedor tem a possibilidade de criar suas aplicações e enviá-las para os servidores que integram a nuvem do Google, podendo usufruir, além da plataforma de desenvolvimento, de sua infraestrutura de armazenamento, processamento e escalabilidade sem a necessidade de manter servidores, com os recursos podendo ser adquiridos da forma IaaS (Infraestrutura com um Serviço) de acordo com as cotas fornecidas pelo serviço à medida que se torne necessário maior poder de processamento, capacidade de tráfego ou armazenamento de dados para seu aplicativo.

Deste modo não se faz necessária a preocupação por parte do cliente com aspectos de configuração e inoperância de servidores, já que estão distribuídos na nuvem de servidores do provedor do serviço, o qual está configurado de tal modo que a falha de uma de suas máquinas não prejudique o funcionamento do resto do sistema. Assim que é realizado o envio do aplicativo, este está pronto para atender aos seus usuários.

Para este propósito o Google Application Engine possui suporte a aplicativos criados com o uso das linguagens de programação Python e Java, e através desta última, várias baseadas na máquina virtual Java. O App Engine permite o uso das

bibliotecas padrões da linguagem, com exceção das que infrinjam as restrições impostas com o grande objetivo de garantir o bom funcionamento do sistema. Para diminuir o efeito negativo das restrições, o Google App Engine procura servir uma série de serviços que facilitam o desenvolvimento e execução de determinadas tarefas.

Os aplicativos criados e disponibilizados a partir do Google App Engine podem ser acessados a partir do seu próprio nome de domínio caso possua uma conta no Google Apps, o qual é um conjunto de aplicativos de caráter corporativo do Google onde é possível associar domínios próprios, ou utilizar o domínio livre padrão *appspot.com* oferecido pelo serviço adicionado ao identificador da aplicação escolhido na hora da sua criação ficando da forma: *identificador.appspot.com*.

O App Engine pode ser utilizado gratuitamente. De acordo com o Google, a capacidade de recursos de armazenamento, CPU e largura de banda disponível para uso gratuito tem o propósito de abrigar de forma satisfatória uma aplicação eficiente de pequeno e médio porte. Tais cotas gratuitas são incrementadas assim que é ativado o faturamento para o aplicativo, sendo pagos os recursos que ultrapassem estas cotas. Através do console de administração do aplicativo é possível gerenciar o orçamento do aplicativo, bem como os recursos consumidos por ele.

3.2 Sandbox

O App Engine, por se tratar de um sistema baseado em *computação em nuvem*, em que o processamento e armazenamento do aplicativo são distribuídos entre vários servidores, necessita de um ambiente virtual seguro para cada aplicativo. Este ambiente chamado de *sandbox* fornece acesso limitado ao sistema operacional, tal limitação possibilita que o Google App Engine distribua as solicitações de web da aplicação entre diversos servidores, podendo iniciar ou interromper os servidores para atender às demandas de tráfego e também que cada aplicação possua uma área isolada segura e confiável independente de hardware,

sistema operacional e localização física do servidor, garantindo que uma aplicação não influencie no funcionamento das demais aplicações. Este método de virtualização além de possibilitar a distribuição na execução do aplicativo, evita o chamado efeito *slashdot*, onde em um ambiente compartilhado, o uso abusivo de recursos por uma aplicação afeta o desempenho das demais.

Para que isso seja possível algumas limitações são impostas ao aplicativo, entre elas podemos citar as mais relevantes segundo o Google:

a) O aplicativo pode acessar outros computadores na internet somente através de solicitações feitas utilizando o protocolo HTTP ou HTTPS por meio dos serviços de obtenção de URL e de e-mail. Outros computadores podem conectar-se à aplicação somente fazendo solicitações HTTP ou HTTPS nas portas padrão;
b) Não é permitido que a aplicação grave no sistema de arquivos. Apenas é permitida a leitura de arquivos enviados juntamente com o código da aplicação. Deve-se utilizar o sistema de armazenamento de dados, cache de memória ou outros serviços do Google App Engine para todos os dados que devam ser persistidos durante as solicitações;
c) É executado o código da aplicação somente em resposta a uma solicitação da web ou uma tarefa programada (*Cron Job*), devendo retornar uma resposta em no máximo 30 segundos em ambos os casos. Não é permitida à aplicação gerar subprocessos (*Threads*) ou executar código após a resposta.

3.3 Armazenamento de dados

Quando se fala em armazenamento de dados estamos tratando de algo vital para o Google Application Engine, pois é o que dá suporte para que este e outros serviços do Google possam ser oferecidos. Vale então destacar algumas tecnologias desenvolvidas pelo Google e que suportam toda sua estrutura de armazenamento de dados, tal como o *Google File System*, a partir da qual se apóiam todas as

subsequentes como o sistema de banco de dados denominado *Bigtable* e como o armazenamento do App Engine está estruturado diante delas.

3.3.1 Google File System

O sistema de armazenamento de dados do App Engine e Google em geral tem sua base apoiada sobre o sistema de arquivos chamado de *Google File System (GFS)*. O desenvolvimento de um novo sistema de arquivos distribuídos veio à tona em 2003 a partir da idéia de armazenar dados de forma escalável, confiável, com alto desempenho e disponibilidade mesmo em máquinas não confiáveis, devendo atender às necessidades de uso do Google de gerar e manter enormes quantidades de dados.

Os arquivos gerenciados por tal sistema geralmente variam de 100 megabytes podendo alcançar vários gigabytes. Para gerenciar o espaço em disco de forma eficiente o GFS organiza os dados em pedaços (*chunks*) – os quais raramente são sobrescritos ou comprimidos – de 64 megabytes, de forma análoga aos blocos – a menor unidade de dados que o sistema é capaz de suportar – dos sistemas de arquivos convencionais, por exemplo, para armazenar um arquivo de 128 megabytes serão utilizados dois pedaços, por outro lado para armazenar um arquivo de um megabyte o GFS deve usar um pedaço de 64 megabytes desperdiçando 63, porém tal caso é tão raro neste sistema de arquivos que não se torna preocupante, sendo mais vantajosa a otimização dos blocos partindo da premissa de que não há arquivos que incitem tal situação.

O GFS foi projetado para rodar de forma otimizada sobre os clusters do Google, onde cada nó do sistema consiste de computadores comuns semelhantes aos computadores domésticos, o que significa que precauções devem ser tomadas a fim de lidar com a grande incidência de falhas e subseqüente perda de dados que podem acarretar. Para contornar tal problema os nós são divididos em dois tipos: um servidor mestre e centenas ou milhares de servidores menores chamados "*Chunkserves*" responsáveis por armazenar os dados.

O servidor mestre é único para todo o sistema de arquivos e não armazena pedaços de dados em si, por sua vez ele contém todos os metadados dos arquivos, entre eles os nomes, permissões e o mapeamento dos seus pedaços e conseqüentes réplicas através de etiquetas de 64, também é monitorado quais processos estão escrevendo ou lendo um pedaço em particular. Todos estes metadados são mantidos atualizados pelo servidor mestre através de mensagens periódicas recebidas dos *Chunkservers*, chamadas de "*Heartbeat messages*", uma analogia à batida do coração. De acordo com apresentações técnicas do Google a organização dos *Chunkservers* se dá por mais de 50 clusters, com milhares de servidores que utilizam o sistema operacional linux por cluster, gerenciando petabytes de dados.

Para ser armazenado o arquivo é quebrado em pedaços de tamanho fixo de 64 megabytes, a cada pedaço o servidor mestre atribui uma etiqueta global única e imutável de 64 bits para o mapeamento lógico dos pedaços que constituem o arquivo, então cada pedaço é replicado várias vezes por toda a rede, com um mínimo de três vezes, porém este número aumenta de acordo com a demanda ou a necessidade de maior redundância.

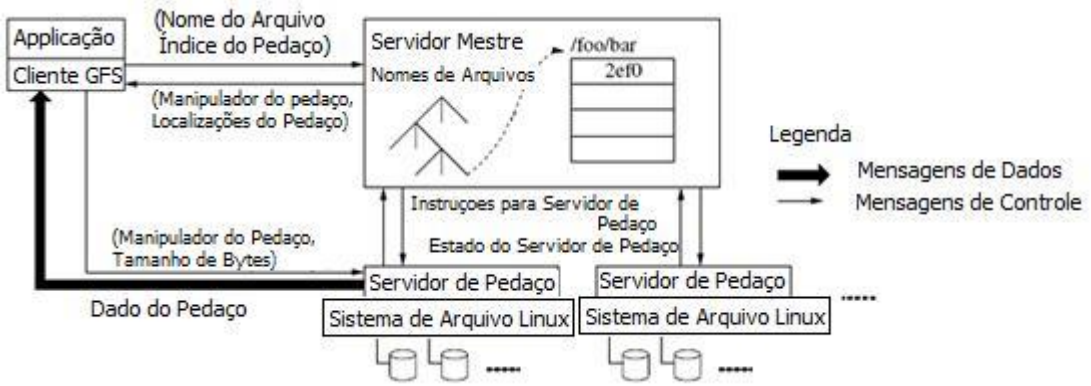

Figura 3.1: Arquitetura do GFS.

3.3.2 BigTable

O BigTable é o sistema de gerenciamento de banco de dados que o Google desenvolveu apoiado sobre o *GFS* a fim de ter um banco de dados distribuído e de grande porte que atendesse ao seus propósitos.

Este sistema foi projetado para rodar de maneira otimizada sobre o *Google File System*, com cada tabela podendo ser subdividida em arquivos de aproximadamente 200 megabytes para serem distribuídas pelo *GFS*. As tabelas funcionam de forma semelhante aos servidores *GFS*, existindo um tabela mestre que serve de referência para as demais.

O desenvolvimento de um sistema sólido e eficiente de banco de dados para rodar sobre a estrutura do Google propiciou o desenvolvimento de várias aplicações por ele fornecidas e o desenvolvimento do Google App Engine.

3.3.3 Armazenamento das entidades

Aqui serão abordados alguns detalhes acerca dos aspectos envolvidos no processo de persistência dos dados dos aplicativos mantidos pelo App Engine mostrando como estes estão estruturados diante do sistema de armazenamento do Google.

O App Engine não utiliza esquemas de modelagem de dados, ou seja, não são criadas tabelas específicas para cada tipo de entidade criada por uma aplicação, contudo utiliza um total de seis “Bigtables” – tabelas mantidas sob o sistema gerenciador de banco de dados do Google – para armazenar todas as entidades e índices, independente do tipo, de todas as aplicações. Tais tabelas estão dispostas de maneira que apenas uma seja usada para armazenar os dados em si e outras cinco para gerenciamento de índices.

3.3.3.1 Entidades

Quando falamos do armazenamento de dados do Google App Engine sempre vem à tona o conceito de entidades. Entidades são estruturas de um determinado tipo semelhantes a linhas de uma tabela em um sistema de banco de dados relacional e são constituídas basicamente por uma chave e um conjunto de propriedades.

O aplicativo utiliza a API de armazenamento de dados para definir modelos de dados e posteriormente criar instâncias destes modelos para serem armazenadas como entidades. Os modelos servem para fornecer uma estrutura comum às entidades e definir o seu tipo.

3.3.3.2 Chaves e grupos de entidades

Cada entidade no armazenamento de dados do App Engine possui uma chave como identificador exclusivo da entidade. Esta chave é exclusiva e serve pra descrever informações de relacionamento pai-filho entre a entidade e outras entidades, tipo da entidade e um nome atribuído pelo aplicativo ou um identificador numérico atribuído pelo armazenamento de dados.

Cada entidade pertence a um grupo de entidade, ou seja, um conjunto de uma ou mais entidades que podem ser manipuladas em uma única transação. Os grupos de entidades informam ao Google App Engine que estas entidades devem ser armazenadas na mesma parte da rede distribuída. Transações servem para garantir a atomicidade das operações aplicando todas as operações caso não haja nenhuma falha, ou não aplicando nenhuma operação caso haja alguma falha.

Ao criar uma entidade, o aplicativo pode atribuir outra entidade como pai desta entidade. Ao atribuir um pai para esta entidade a coloca no mesmo grupo de entidades do pai.

Uma entidade sem pai é chamada de entidade raiz. Uma entidade que é pai de outra também pode ter um pai. Uma cadeia de entidades pai, desde a entidade até a raiz é o caminho da entidade, e os membros do caminho são os ancestrais da

entidade. O pai da entidade é atribuído na hora de criação da entidade e não pode mais ser modificado. Caso uma entidade ancestral seja excluída a entidade descendente não é excluída.

Portanto todas as entidades com a mesma entidade raiz como ancestral estão no mesmo grupo de entidades. O App Engine busca guardar todas as entidades pertencentes a um mesmo grupo no mesmo nó dos servidores de armazenamento de dados. Logo, quanto mais grupos de entidades o aplicativo possuir, mais eficiente será a distribuição das entidades entre os nós do armazenamento de dados, melhorando o desempenho da criação e atualização de dados.

3.3.3.3 Propriedades

Propriedades são os dados que dão razão às entidades, armazenando os atributos definidos no modelo da entidade. Para armazenar as propriedades o App Engine utiliza cinco tipos de valores básicos, podendo haver propriedades que utilizam uma composição de dois ou mais destes tipos:

a) *str*: cadeia de caracteres de tamanho variável. O tamanho físico – número total de bytes necessários – ocupado depende do tamanho da cadeia e do tipo de codificação utilizado. Internamente valores do tipo *str* utilizam a codificação UTF-8.
b) *int32*: campos deste tipo codificam valores inteiros utilizando 32 bits, portanto utilizam 4 bytes de espaço físico.
c) *int64*: campos deste tipo codificam valores inteiros utilizando 64 bits, portanto utilizam 8 bytes de espaço físico.
d) *double*: representa números em pontos flutuantes utilizando 64 bits, consumindo 8 bytes de espaço físico.
e) *bool*: armazena os dois valores – *true* ou *false* –, utilizando 4 bytes de espaço físico. Apesar de poderem ser armazenadas de forma mais eficiente utilizando apenas um byte, uma codificação em 4 bytes é necessária a utilização do *protocol buffers*.

Tabela 3.1: Espaço físico gasto por cada tipo básico de propriedade.

Tipo	Bytes
str	*
int32	4
int64	8
double	8
bool	4

3.3.3.4 Tabela de entidades

A tabela de entidades é a parte fundamental do sistema de armazenamento oferecido pelo Google Application Engine, pois é ela que comporta todas as entidades criadas por todas as aplicações sob o suporte do App Engine independente de que tipos sejam essas entidades. Cabe a esta tabela armazenar cada entidade de fato através de todos os dados necessários para sua descrição bem como os dados de suas propriedades.

Para cada entidade existe uma chave composta pelo identificador da aplicação que a criou e um caminho. O identificador da aplicação é o mesmo utilizado na hora da sua criação e utilizado para acessá-la através do domínio padrão *appspot.com*. Devido ao modo como estes identificadores são codificados e armazenados na *Bigtable*, escolher um identificador pequeno geralmente não resulta em uma economia drástica de espaço, porém podem ocorrer pequenos ganhos de espaço de acordo com o número de entidades regidas pelo aplicativo. Entretanto um identificador pequeno pode surtir em ganhos de espaço caso a aplicação possua em suas entidades muitas propriedades que referenciem outras entidades já que neste caso o identificador do aplicativo não é codificado. O caminho descreve relacionamentos com outras entidades, o tipo da entidade e um nome atribuído pelo aplicativo ou um identificador numérico atribuído pelo armazenamento de dados à entidade.

Além dos metadados a respeito da entidade que estão contidos na sua chave, cada entidade possui duas colunas para este fim. Uma serve para definir o grupo da entidade e funciona contendo a chave de uma entidade raiz, a qual pode ser ela mesma. A outra é formada por uma cadeia de caracteres que contém o tipo da entidade à que a entidade referenciada na coluna anterior pertence.

Cada entidade possui uma ou mais propriedades. Cabe também a esta tabela reter as informações referentes a estas propriedades. Estas informações são compostas por um nome e um valor que corresponde a um tipo básico ou um tipo definido através da classe *Property*.

O total de espaço consumido por uma entidade não depende apenas dos tipos e valores das propriedades, mas também de seus nomes. Por se tratar de armazenamento de dados essencialmente sem o suporte de esquemas, cada entidade de um tipo em particular deve armazenar o nome de cada propriedade além do seu valor, mesmo que todas as outras entidades do tipo usem o mesmo conjunto de propriedades. Embora esta redundância resulte em um leve "*overhead"* de armazenamento, ela oferece uma maior flexibilidade na modelagem das entidades, já que uma entidade não precisa de fato conter todas as propriedades definidas no seu modelo.

Internamente o App Engine armazena as entidades utilizando um método chamado *protocol buffers*. Trata-se de um eficiente mecanismo de codificação e serialização de dados estruturados utilizado pelo Google para quase todos os seus que necessitam de tal abordagem. Ao invés de armazenar cada propriedade em uma coluna individual na linha correspondente, uma única coluna contendo um código binário gerado pela aplicação do *protocol buffers* é utilizada para armazenar todas as propriedades. O *protocol buffers* associa um número que requer no mínimo um byte de espaço para cada propriedade. Portanto além dos bytes necessários para o valor das propriedades deve-se considerar o espaço extra gerado pelo *protocol buffers*.

Por fim esta tabela também armazena dados para índices personalizados, necessários para executar *queries* complexas. Para cada índice personalizado é armazenado um identificador de 64 bits. Um campo armazena a chave da entidade e de seus ancestrais, caso ela não seja uma entidade raiz. Por fim são armazenadas as propriedades referentes ao índice personalizado. Assim como as propriedades da entidade, cada dado do índice personalizado é codificado utilizando o *protocol buffers* e armazenado em uma única coluna.

3.3.3.5 Tabelas de índices

Quatro tabelas são usadas para armazenar todos os índices para cada aplicação do App Engine. Cada uma destas tabelas armazena dados para um tipo de índex em particular. As primeiras três destas tabelas – *EntitiesByKind*, *EntitiesByProperty ASC* e *EntitiesByProperty DESC* – são gerenciadas automaticamente pelo App Engine assim que novas entidades são escritas. A quarta, EntitiesByCompositeProperty, deve ser definida explicitamente.

A tabela *EntitiesByKind* possibilita recuperar todas as entidades de um tipo em particular. Toda vez que uma entidade é criada e adicionada um registro nesta tabela referente à entidade é adicionado possibilitando-a de ser requisitada posteriormente. Esta tabela contém o identificador da aplicação, uma cadeia de caracteres identificando o tipo da entidade e por fim a chave da entidade.

A não ser que seja definido explicitamente no seu modelo, cada propriedade da entidade também é indexada automaticamente, com exceção de propriedades do tipo *Text* e *Blob*, que não podem ser indexadas. Estes índices não podem ser vistos no arquivo de configuração de índices ou no painel de configuração do aplicativo, mas eles existem para facilitar requisições condicionadas a uma propriedade. Cabe as tabelas *EntitiesByProperty ASC* e *EntitiesByProperty DESC* armazenar estes índices. Elas possuem os mesmos campos da tabela descrita anteriormente, com a adição de um campo para o nome da propriedade e outro para o seu valor. Ou seja, a entidade possui um registro nesta tabela para cada valor da propriedade indexada.

Se uma propriedade possui vários valores vai precisar de várias linhas nesta tabela para armazená-la.

As tabelas de índices descritas anteriormente suportam índices automáticos, mas índices personalizados são necessários para executar requisições mais complexas, como as com múltiplos comandos de ordenação. Estes índices são modelados com o auxílio de um arquivo específico para este propósito, enviado juntamente com a aplicação.

Para o armazenamento de índices personalizados o App Engine conta com duas tabelas. Uma tabela existe para armazenar a definição dos índices personalizados. Com a finalidade de definir o índice, esta tabela contém o tipo da entidade afetada pelo índice, nomes das propriedades por ele referenciadas, valores *booleanos* para cada propriedade a fim de indicar o rumo da sua ordenação e um campo *booleano* para indicar se o índice usa filtros de ancestral. Esta tabela ainda contém metadados adicionais contendo o identificador do índice, identificador da aplicação e uma coluna com um valor inteiro para definir o estado do índice, o qual pode ser: construindo, servindo, apagando ou erro.

A tabela *EntitiesByCompositeProperty* é a tabela de índices personalizados que armazena dados das entidades cobertas por tais índices. Esta tabela possui o identificador do índice, identificador da aplicação, tipo da entidade abrangida pelo índice, chave do ancestral – caso o índice use filtro de ancestral –, valor das propriedades indexadas e chave da entidade. Deve-se tomar cuidado com os chamados índices explosivos, quando índices personalizados que referenciam diversas propriedades com diversos valores podem ficar muito grandes com alguns poucos valores, pois a tabela deve armazenar uma linha para cada permutação dos valores de cada propriedade indexada.

3.3.3.6 Tabela de seqüência de identificadores

Por fim, além das seis tabelas para armazenamento das entidades e respectivos índices, o Google App Engine possui uma tabela extra usada para gerar identificadores numéricos tanto para entidades como para índices.

Cada aplicação possui no mínimo dois registros nesta tabela, uma para todas as entidades raiz e outra para a definição de todos os índices personalizados, podendo aumentar este número em uma linha por grupo de entidade que possui ou já possuiu uma entidade não raiz. Portanto caso a entidade possua apenas entidades raízes serão necessárias apenas duas linhas nesta tabela.

Os registros desta tabela estão dispostos em colunas que carregam o identificador da aplicação, chave da entidade – no caso de ser uma seqüência para um grupo de entidades, do contrário usam-se valores especiais –, e por fim o próximo identificador numérico a ser usado.

3.3.4 Limites

O Google App Engine impõe os seguintes limites ao seu sistema de armazenamento de dados:

a) Cada entidade deve possuir um tamanho máximo de 1 megabyte.
b) O número máximo de valores em um índice de uma entidade, ou seja, o número máximo de linhas que a entidade pode ter em uma tabela de índice é de 1000 valores. Isso acontece para evitar que uma atualização da entidade demore muito tempo, já que todos os valores referentes à propriedade devem ser atualizados também na tabela de índices.
c) O número máximo de entidades armazenadas ou excluídas em lote numa única operação (*batch put* e *batch delete*) deve ser de 500 entidades.
d) É suportado um número máximo de 1000 entidades restauradas em lote (*batch get*).

e) O número máximo de entidades devolvidas por uma consulta é de 1000 entidades.

3.4 Serviços do Google App Engine

O Google Application Engine conta com diversos serviços que facilitam a execução de determinadas operações por parte da aplicação. Para tanto, todas as linguagens de programação suportadas pelo App Engine oferecem APIs específicas para cada serviço em questão, descomplicando a execução de tarefas a eles relacionadas.

3.4.1 Contas do Google

Este serviço é responsável pela integração entre o aplicativo desenvolvido utilizando o App Engine com as contas do Google. Permite elaborar um sistema de *login* no aplicativo para que o usuário se autentique com o uso de sua conta do Google. Torna mais ágil o uso do aplicativo já que o usuário talvez não precise criar uma nova conta e economizando esforço na implementação de um sistema de contas específico para o aplicativo.

O aplicativo pode detectar através deste serviço se um usuário fez login no aplicativo utilizando uma conta do Google. Pode também redirecionar o usuário à página das contas do Google para que ele efetue login ou crie uma nova conta. Do mesmo modo fornece um método para que o usuário saia do aplicativo.

Este serviço possibilita ao aplicativo ter disponível o endereço de e-mail do usuário e o nome de exibição a ele associado. Também o torna capaz de detectar se o usuário atual é um administrador do aplicativo cadastrado no console de administração.

O objeto que representa o usuário pode ser armazenado no sistema de armazenamento de dados do App Engine como um tipo de dado especial, porém não se comporta como um identificador estável, ou seja, se o usuário modificar seu endereço de e-mail, o valor do dado armazenado não fará mais referência a um usuário válido.

3.4.2 Obtenção de URL

A partir deste serviço a aplicação é capaz de acessar recursos na internet, como serviços da web e outras aplicações, emitindo solicitações *HTTP* ou *HTTPS* e recebendo respostas. Para tanto utiliza a mesma infraestrutura de alta velocidade do Google.

A URL a ser obtida deve usar as portas padrão para HTTP (80) e HTTPS (443) e pode utilizar qualquer um dos seguintes métodos HTTP: *GET, POST, PUT, HEAD e DELETE*. A solicitação pode incluir cabeçalhos e uma carga útil (corpo).

Para impedir que um aplicativo cause uma recursão infinita de solicitações, o código responsável por tratar a solicitação não tem permissão para obter seu próprio URL, mesmo assim deve-se ficar atento, pois é possível causar recursão infinita por outros meios. Uma chamada do serviço de obtenção de URL é síncrona e não retornará até que o serviço receba uma resposta do host remoto.

O aplicativo pode obter URL com o método *HTTPS* para se conectar a servidores seguros, com os dados da solicitação e da resposta transmitidos pela rede de forma cifrada. O Proxy utilizado pelo serviço de obtenção de URL não pode autenticar o host com o qual está entrando em contato por não possuir uma cadeia de confiança de certificados, aceitando, portanto todos os certificados, incluindo os auto-assinados. O serviço também não é capaz de detectar ataques de interceptação entre o App Engine e o host solicitado.

O aplicativo pode definir cabeçalhos HTTP para a solicitação de saída, com exceção de alguns que por motivo de segurança não podem ser modificados:

a) Content-Length
b) Host
c) Referer
d) Vary
e) Via
f) X-Fowarded-For

Os cabeçalhos acima são definidos precisamente pelo Google App Engine, conforme apropriado.

As respostas das solicitações incluem código, cabeçalho e corpo. Caso os dados da resposta possuam código de redirecionamento, o serviço seguirá esse redirecionamento. O serviço seguirá um máximo de 5 respostas de redirecionamento, havendo também a opção de não seguir estes redirecionamentos, apenas retornando uma resposta de redirecionamento para o aplicativo. Por padrão se a resposta exceder o limite de tamanho de resposta imposto pelo App Engine ela ficará truncada.

Este serviço está disponível também no servidor de desenvolvimento, o qual entra em contato com os hosts diretamente do computador do desenvolvedor.

Os seguintes limites são aplicados ao serviço de obtenção de URL do App Engine:

a) O máximo de dados enviados em uma solicitação é de 1 megabyte.
b) O máximo de dados recebidos como resposta a uma solicitação é de 1 megabyte.

3.4.3 Cache de memória

O App Engine fornece o serviço de cache de memória de alto desempenho capaz de ser compartilhado por diversas instâncias do aplicativo. Este tipo de serviço é extremamente útil para dados temporários que não necessitam ser persistidos no sistema de armazenamento de dados.

Uma situação que é boa candidata para o uso do serviço de cache de memória é quando muitas solicitações do arquivo fizerem referência ao mesmo conjunto de dados. O sistema pode requerer os dados junto ao sistema de armazenamento e armazená-los em cache para solicitações subsequentes, voltando a solicitar os serviços do armazenamento de dados apenas quando as informações contidas no serviço de cache expirarem. Este serviço pode ser extremamente útil, no entanto deve-se considerar o fato que os valores contidos em cache podem expirar a qualquer momento mesmo antes do prazo definido para isso.

Os valores armazenados em cache são retidos o máximo possível, mas podem ser removidos quando um novo valor é adicionado a ele e há pouca memória disponível. Quando isso acontece os valores menos usados recentemente são removidos primeiro. O aplicativo pode fornecer um prazo para que o dado expire, a tendência é que o valor seja removido exatamente ao final do prazo, porém pode ser removido por outras razões. Em circunstâncias raras os valores podem desaparecer do cache de memória antes da data de expiração por razões diferentes das de pressão de memória citada acima. Isso ocorre porque, embora o sistema de cache seja resistente às falhas de servidor, os valores nele contidos não são salvos em disco e uma falha no servidor pode fazer com que o dado se perca. Essa situação torna evidente que uma aplicação não deve esperar que um valor em cache esteja sempre disponível.

Há também um limite para um valor armazenado em cache, que é de 1 megabyte.

3.4.4 Mensagens

Os aplicativos são capazes de enviar mensagens de e-mail facilmente fazendo uso da infraestrutura do Google através do serviço de mensagens fornecido pelo App Engine.

Com o uso deste serviço, a aplicação pode enviar mensagens de e-mail para um ou mais destinatários. A mensagem contém assunto, corpo de texto sem formatação e um corpo em *HTML* opcional, também é possível conter arquivos anexados.

Por questões de segurança, o aplicativo só pode enviar mensagens com o endereço de remetente sendo o de um administrador do aplicativo ou o endereço de e-mail da conta do Google do usuário atual conectado. É permitido qualquer endereço de e-mail para o destinatário que deve estar nos campos “para”, “cc” ou “cco”.

É permitido que o aplicativo envie mensagens com arquivos anexados a ela. Por questões de segurança um conjunto restrito de tipos de arquivo é aceito e sua extensão deve corresponder a estes tipos. Veja abaixo a lista de tipos MIME (sigla em inglês para Extensões Multi função para Mensagens de Internet) e suas respectivas extensões de nomes permitidos.

Tabela 3.2: Tipos MIME permitidos para arquivos anexados.

Tipo MIME	Extensões de nome de arquivo
image/x-ms-bmp	bmp
text/css	css
text/comma-separated-values	csv
image/gif	gif
text/html	htm html
image/jpeg	jpeg jpg jpe
application/pdf	pdf
image/png	png
application/rss+xml	rss
text/plain	text txt asc diff pot

image/tiff	tiff tif
image/vnd.wap.wbmp	wbmp
text/calendar	ics
text/x-vcard	vcf

Quando o serviço de envio de mensagens é solicitado, ele coloca a mensagem em uma fila e a chamada retorna imediatamente para o aplicativo. O serviço usa procedimentos padrões para contatar cada servidor de e-mail do destinatário, entregando a mensagem ou repetindo a tentativa caso não seja possível entrar em contato com o servidor. Caso o serviço não possa entregar a mensagem uma mensagem de erro é enviada para o endereço do remetente. O aplicativo não recebe nenhuma notificação sobre êxito ou falha na entrega da mensagem.

O serviço de mensagens do App Engine possui os seguintes limites estipulados:

a) 1 megabyte de tamanho máximo de mensagem, incluindo anexos.
b) 16 quilobytes de tamanho máximo de mensagem quando o destinatário for um administrador do aplicativo.

3.4.5 Manipulação de imagens

Através deste serviço o aplicativo é capaz de manipulas imagens. Este serviço é capaz de redimensionar, girar, inverter e cortar imagens.

O serviço de manipulação de imagens aceita dados de imagem nos formatos *JPEG*, *PNG*, *GIF* (incluindo animados), *BMP*, *TIFF* e *ICO*. São retornadas imagens transformadas no formato *JPEG* ou *PNG*. Caso o formato de entrada for diferente do formato de saída, o serviço se encarrega de realizar a conversão.

O tamanho máximo dos dados, tanto enviados como recebidos do serviço de manipulação de imagens é de 1 megabyte.

3.4.6 XMPP

Este serviço possibilita que o aplicativo envie ou receba mensagens instantâneas de qualquer serviço de mensagens instantâneas compatível com o protocolo XMPP, como o Google Talk. Assim uma aplicação pode enviar ou receber mensagens de bate-papo de usuários de algum servidor XMPP ou até de outras aplicações do App Engine, enviar convites de bate-papo ou requisitar informações do estado do usuário. As mensagens recebidas são tratadas de forma similar as requisições web.

Algumas possibilidades deste serviço incluem a criação de um usuário de chat automatizado, o chamado "*chat bot*", notificações instantâneas ou um meio de comunicação entre aplicações do App Engine.

Para que a aplicação possa receber mensagens ela deve possuir um endereço, ou "*JID*". Um tipo de endereço que a aplicação possui é formado pelo identificador da aplicação e o domínio *appspot.com*, da seguinte maneira: *appid@appspot.com*. Uma alternativa se dá da seguinte forma: *nome@app-id.appspotchat.com*. Onde o *nome* pode ser qualquer coisa contendo letras, números e hífens e *app-id* é o identificador do aplicativo. Todas as mensagens enviadas para qualquer endereço no formato suportado é encaminhada para a aplicação na sua versão padrão. As mensagens podem ser endereçadas para outras versões da aplicação (caso existam) pelo seguinte endereço: nome@version.latest.app-id.appspotchat.com.

Uma aplicação precisa mais que o seu endereço de bate-papo para que possa efetivamente receber mensagens. É necessário que este serviço seja ativado através do arquivo de configuração da aplicação e também possuir um tratador para

as solicitações oriundas deste serviço no endereço: */_ah/xmpp/message/chat/*. O tratador deve ser capaz de tratar solicitações HTTP que usam o método *POST*.

Este serviço está limitado ao tamanho máximo de 100 quilobytes, tanto para as mensagens enviadas, quanto para as mensagens recebidas.

3.4.7 Task Queue

Este serviço está disponível sob caráter experimental, estando ainda em desenvolvimento. A partir deste serviço as aplicações podem executar tarefas fora da requisição de um usuário, porém iniciada pela requisição. Se uma aplicação precisa executar algum trabalho de fundo, ela organiza este trabalho em pequenas tarefas e as coloca em uma ou mais filas. O App Engine então se encarrega de automaticamente detectar tais tarefas e executá-las assim que os recursos do sistema permitam.

Para ser executada, uma tarefa necessita de dados e uma URL relativa, chamada de trabalhador para a tarefa, para a qual os dados são enviados. É possível dar também um nome opcional para a fila. A URL do trabalhador da tarefa é passada para o construtor da tarefa, caso a URL não seja especificada é usada uma URL padrão como seu trabalhador: */_ah/queue/nome_da_fila*.

O App Engine providencia uma lista padrão com o nome de *default* para todos os aplicativos, a qual não necessita de nenhuma configuração para ser usada e possui uma vazão de 5 tarefas por segundo. As filas tendem a se comportar do modo FIFO (*first in, first out*), entretanto múltiplas tarefas de uma fila podem ser executas simultaneamente pelo sistema.

As tarefas funcionam da mesma maneira que requisições web, o que significa que estão condicionadas às mesmas imposições feitas pelo App Engine, como o tempo de execução limitado a 30 segundos. Se a execução de uma tarefa em particular falhar, retornando um código HTTP entre 200 e 299, o App Engine tentará executá-la até obter sucesso, com o tempo entre as tentativas aumentando gradativa

até alcançar o mínimo de uma tentativa por dia. Ao criar o código da tarefa deve-se considerar a possibilidade excepcional da tarefa ser executada mais de uma vez, garantindo que a execução repetida da tarefa não causará danos. É possível proteger a tarefa configurando para que seja executada apenas pelo administrador prevenindo que usuários externos maliciosos tenham acesso direto a ela através da sua URL.

É possível associar a tarefa a transações, fazendo que a tarefa seja adicionada à fila se a transação for cometida com sucesso, fazendo-a parte da transação, tendo o mesmo nível de isolamento e consistência. Um aplicativo não pode inserir mais que 5 tarefas transacionais em uma única transação.

O serviço Task Queue está sujeito às seguintes limitações:

a) O tamanho máximo do objeto da tarefa deve ser de 10 quilobytes.
b) O número máximo de 10 filas ativas, sem incluir a fila padrão.
c) Máximo de 50 chamadas de tarefas por segundo.
d) A tarefa deve ser executada em no máximo 30 dias.
e) O máximo de 100 tarefas inseridas em lote.

3.4.8 Blobstore

Assim como o serviço anterior, este serviço está em desenvolvimento, ainda sob caráter experimental. O serviço Blobstore permite à aplicação servir objetos de dados, chamados de *Blobs*, que são muito maiores que os tamanhos permitidos para objetos no serviço de armazenamento de dados. Os *Blobs* definidos por este seviço não possuem relação com o tipo de propriedade *Blob* do serviço de armazenamento de dados. *Blobs* são criados pelo envio de um arquivo através de uma requisição HTTP, tipicamente fazendo isso por um formulário com um campo para envio de arquivos. Quando o formulário é enviado, o serviço Blobstore cria um *blob* do arquivo e disponibiliza uma referência a ele, chamada de chave do *blob*, a qual pode ser usada para servi-lo posteriormente.

A partir deste serviço a aplicação pode servir objetos de dados com até 50 megabytes de tamanho. A aplicação não cria os dados destes objetos diretamente, mas cria *blobs* indiretamente pelo envio do formulário web. *Blobs* não podem ser modificados depois de criados, mas podem ser excluídos.

O serviço de manipulação de imagens pode se integrar com o serviço Blobstore tendo um de seus valores como fonte.

Os seguintes limites se aplicam ao serviço Bloobstore:

a) 50 megabytes como tamanho máximo para os objetos armazenados.
b) 1 megabytes de tamanho máximo de dados que podem ser lidos do Blobstore através de uma chamada da API.

3.5 Cotas e faturamento

A criação e uso de um aplicativo utilizando o App Engine é gratuito, entretanto o uso está condicionado ao consumo de certa quantidade máxima de recursos chamada de cotas. O sistema de cotas do App Engine auxilia o administrador da aplicação no controle do seu orçamento despendendo recursos de acordo com as necessidades. Um dos limites impostos pelo App Engine está relacionado ao número máximo de 10 aplicativos por conta do desenvolvedor.

3.5.1 Cotas faturáveis e cotas fixas

Os recursos do App Engine são medidos mediante dois tipos de cotas: uma cota faturável e uma cota fixa.

As cotas faturáveis são os valores máximos definidos pelo administrador da aplicação ou impostos pelos limites gratuitos do App Engine, ou seja, recursos extras

que podem ser comprados pelo administrador do aplicativo. Cada aplicativo recebe uma cota faturável gratuitamente, podendo ser aumentada com a ativação do faturamento, definindo um orçamento diário e em seguida distribuindo o orçamento para as cotas através do painel de administração. O Valor cobrado será apenas pelos recursos que o aplicativo realmente usa acima dos estipulados pela cota gratuita.

Cotas fixas são valores máximos de recursos pré-definidos pelo Google App Engine a fim de garantir a integridade do sistema. Eles descrevem os limites da arquitetura nos quais todos os aplicativos devem ser executados e existem para garantir que um aplicativo não consuma demasiados recursos, a ponto de afetar o desempenho dos outros aplicativos executados no sistema.

Uma situação em que os limites determinados pelas cotas fixas se fazem presente é o limite de 30 segundos que aplicação possui para emitir uma resposta a uma solicitação da web. Assim se o aplicativo demorar muito para emitir a resposta, o processo referente a ele será encerrado e o servidor retornará um código de erro ao usuário. Outra situação é o número máximo de 1000 resultados que uma consulta ao armazenamento de dados pode retornar. A finalidade desta restrição é poupar recursos do armazenamento de dados.

Ao se ativar o faturamento para o aplicativo algumas cotas fixas do aplicativo são automaticamente incrementadas.

3.5.2 Renovação de recursos

O Google Application Engine registra o uso diário de recursos e os considera esgotados quando a quantidade utilizada atinge a cota estabelecida. Um dia é um período de 24 horas, começando à meia-noite pelo horário do pacífico (GMT -08:00). Ao começo de cada dia todas as medições referentes ao uso de recursos são reiniciadas com exceção do armazenamento de dados que sempre representam a atual quantidade de espaço utilizado pelo armazenamento de dados.

3.5.3 Cotas por minuto

O App Engine limita o consumo de recursos no intervalo de um minuto utilizando um sistema de cotas fixas chamado de cotas por minuto, a fim de evitar que o aplicativo consuma toda sua cota em um período muito curto de tempo e impede que um aplicativo afete o funcionamento de outro ao monopolizar um determinado recurso.

Caso um recurso esgote seus limites por minuto haverá uma notificação no console de administração da aplicação e as solicitações que utilizam tal recurso serão negadas.

O nível de recursos disponíveis por minuto é aumentado automaticamente apenas mediante a habilitação do faturamento.

3.5.4 Esgotamento de recursos

Os aplicativos possuem uma série de recursos sujeitos às cotas com quantidades alocadas para um período de 24 horas. Quando todo o recurso é consumido, este fica indisponível até a renovação da cota, implicando que o aplicativo não funcionará corretamente até lá. Quando um recurso necessário para iniciar uma solicitação está esgotado o App Engine retorna um código de status HTTP 403, que significa acesso proibido e não executa a chamada. Este comportamento específico se dá com o esgotamento dos seguintes recursos:

a) Solicitações;
b) Tempo de CPU;
c) Largura de banda (entrada e saída).

Quando algum outro recurso estiver esgotado, como por exemplo, o serviço de mensagens, as tentativas de consumo do recurso irão gerar uma exceção específica.

3.5.5 Recursos de solicitações

Esta seção descreve os recursos cotáveis de solicitações. Estes são os recursos mais utilizados pelo aplicativo e são consumidos quando uma solicitação web é feita ao aplicativo. Alguns recursos de solicitações também podem ser contabilizados juntamente com outros recursos que não estão diretamente relacionados a solicitações, como ocorre com certos recursos de armazenamento de dados ou do serviço de mensagens.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.3: Cotas para recursos de solicitações.

Recurso	**Cota padrão gratuita**		**Cota com faturamento ativado**	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Solicitações	1.300.000 solicitações	7.400 solicitações/minuto	43.000.000 solicitações	30.000 solicitações/minuto
Largura de banda de saída	1 gigabyte	56 megabytes/minuto	1 gigabyte gratuitos; máximo de 1.046 GB	10 gigabytes/minuto
Largura de banda de entrada	1 gigabyte	56 megabytes/minuto	1 gigabyte gratuitos; máximo de 1.046 GB	10 gigabytes/minuto
Tempo de CPU	6,5 horas de CPU	15 minutos de CPU/minuto	6,5 horas de CPU gratuito, máximo de 1.729 horas de CPU	72 minutos de CPU/minuto

É importante ressaltar que solicitações feitas por meios seguros, ou seja, utilizando o protocolo *HTTPS* consomem maior tempo de CPU e largura de banda

do que solicitações normais devido à sobrecarga que ela acarreta, tornando-a conseqüentemente mais cara e lenta.

3.5.5.1 Solicitações

Este recurso representa o número total de solicitações ao aplicativo e é consumido sempre que uma solicitação ao aplicativo é feito. Com o faturamento ativado as cotas para este recurso permitem 500 solicitações por segundo. Este número pode ser aumentado mediante ao pedido através de um formulário especial caso a aplicação exija.

3.5.5.2 Largura de banda de saída

Largura de banda de saída é um recurso faturável que representa a quantidade de dados enviados em resposta às solicitações. Inclui tanto os dados enviados em solicitações seguras como em solicitações não seguras, enviados em mensagens de e-mail e dados enviados em solicitações HTTP através do serviço de obtenção de URL.

3.5.5.3 Largura de banda de entrada

Semelhante ao recurso anterior, porém é um recurso faturável que representa a quantidade de dados recebidos das solicitações ao aplicativo. Inclui tanto os dados recebidos em solicitações seguras como em solicitações não seguras e dados recebidos em resposta às solicitações HTTP do serviço de obtenção de URL.

3.5.5.4 Tempo de CPU

Este recurso representa o tempo total de processamento efetivo despendido pelo App Engine no processamento de manipulação de solicitações ou acessos ao armazenamento de dados.

Este é um item que não se encontra muito claro ao entendimento em relação ao modo como é calculado. Segundo o Google, o Tempo de CPU é medido registrando o número de ciclos de CPU utilizados para o processamento, com 1 segundo de Tempo de CPU sendo capaz de realizar 1200 ciclos, equivalente ao número de ciclos que um processador Intel x86 de 1,2 GHZ pode realizar no mesmo 1 segundo.

3.5.6 Recursos de armazenamento de dados

Estas cotas contabilizam os recursos utilizados pelo sistema de armazenamento de dados quando se faz acesso à API de armazenamento de dados ou o total de dados armazenados pela aplicação em todos os serviços.

As tabelas abaixo mostram os limites diários e por minuto para situações com e sem o faturamento ativado para os recursos de armazenamento de dados:

Tabela 3.4: Cotas para recursos de armazenamento de dados.

Recurso	**Cota padrão gratuita**	**Cota com faturamento ativado**
Dados armazenados	1 gigabyte	1 gigabyte gratuito; sem máximo
Número de índices	100	200

Tabela 3.5: Cotas para recursos de armazenamento de dados.

Recurso	**Cota padrão gratuita**		**Cota com faturamento ativado**	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API de armazenamento de dados	10.000.000 chamadas	57.000 chamadas/minuto	140.000.000 chamadas	129.000 chamadas/minuto
Queries	10.000.000 queries	57.000 queries/minuto	200.000.000 queries	129.000 queries/minutos
Dados enviados à API	12 gigabytes	68 megabytes/minuto	72 gigabytes	153 megabytes/minuto

Dados recebidos da API	115 gigabytes	659 megabytes/minuto	695 gigabytes	1.484 megabytes/minuto
Tempo de CPU do armazenamento de dados	60 horas de CPU	20 minutos de CPU/minuto	1.200 horas de CPU	50 minutos de CPU/minuto

3.5.6.1 Chamadas da API de armazenamento de dados

Este recurso representa qualquer chamada à API do armazenamento de dados. Estes recursos incluem recuperar, criar, atualizar ou excluir uma entidade ou executar uma consulta.

3.5.6.2 Dados armazenados

Este é o único recurso faturável relativo ao serviço de armazenamento e representa a quantidade total de dados de entidades e índices do aplicativo armazenados no sistema de armazenamento de dados, além dos dados na memória cache e no serviço Blobstore. O espaço utilizado por entidades e seus índices se dá como descrito na seção que aborda o sistema de armazenamento de dados.

Este é o único recurso faturável que não tem seus valores contabilizados reiniciados com o início da nova diária. O valor deste recurso representa a quantidade atual de dados armazenados pelo aplicativo no sistema de armazenamento de dados.

3.5.6.3 Dados enviados à API

Os dados enviados à API representam a quantidade de dados enviados ao sistema de armazenamento de dados através de chamada da sua API. Contam para este recurso o envio de dados para a criação ou atualização de uma entidade ou dados necessário para a realização de uma consulta.

3.5.6.4 Dados recebidos da API

Os dados recebidos da API representam a quantidade de dados recebidos do sistema de armazenamento de dados através da chamada da sua API. Contam para este recurso os dados retornados da recuperação de uma entidade ou da realização de uma consulta.

3.5.6.5 Tempo de CPU do armazenamento de dados

Limite relativo ao tempo de CPU gasto realizando operações do armazenamento de dados. Estes gastos também são computados para a cota de tempo de CPU, portanto é calculado da mesma maneira.

3.5.6.6 Queries

Representa o número de vezes que o aplicativo executou uma query no armazenamento de dados. É importante notificar que algumas operações (*IN* e *!=*) excutam múltiplas queries internamente contando para esta cota.

3.5.6.7 Número de índices

Contabiliza o número de índices existentes para a aplicação. Isto inclui índices que foram criados, mas não aparecem na configuração da aplicação por não terem sido excluídos com o comando *vacuum_indexes*.

3.5.7 Recursos de mensagens

Os recursos de mensagens são todos os recursos envolvidos com o serviço de mensagens do App Engine.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.6: Cotas para recursos do serviço de mensagens.

Recurso	Cota padrão gratuita		Cota com faturamento ativado	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API de mensagens	7.000 chamadas	32 chamadas/minuto	1.700.000 chamadas	4.900 chamadas/minuto
Destinatários de e-mail	2.000 destinatários	8 destinatários/minuto	2.000 destinatários gratuito; máximo de 7.400.000 destinatários	5.100 destinatários/minuto
E-mail para administradores	5.000 e-mails	24 e-mails/minuto	3.000.000 e-mails	9.700 e-mails/minuto
Dados enviados no corpo da mensagem	60 megabytes	340 kilobytes/minuto	29 gigabytes	84 megabytes/minuto
Anexos enviados	2.000 anexos	8 anexos/minuto	2.900.000 anexos	8.100 anexos/minuto
Dados enviados como anexos	100 megabytes	560 kilobytes/minuto	100 gigabytes	300 megabytes/minuto

3.5.7.1 Chamadas da API de mensagens

Recurso contabilizado sempre que o aplicativo faz uma chamada à API do serviço de mensagens como na operação de enviar uma mensagem de e-mail.

3.5.7.2 Destinatário de e-mail

Trata-se do único recurso faturável relativo aos serviços de mensagens do App Engine. Determina efetivamente o número máximo de e-mails que o aplicativo pode mandar para destinatários que não estão cadastrados como administradores do sistema.

3.5.7.3 E-mail para administradores

Recurso semelhante ao anterior com o diferencial de não ser faturável e de contabilizar apenas os e-mails enviados pelo aplicativo para seus administradores.

3.5.7.4 Dados enviados no corpo da mensagem

Contabiliza os dados que são enviados no corpo das mensagens de e-mail. Estes dados também são contabilizados para a cota de Largura de banda de saída.

3.5.7.5 Anexos enviados

Serve para contabilizar o número total de anexos que são enviados juntamente com as mensagens de e-mail.

3.5.7.6 Dados enviados como anexo

Refere-se à quantidade de dados enviados como anexos juntamente com as mensagens de e-mails. Estes dados também são contabilizados para a cota de Largura de banda de saída.

3.5.8 Recursos de obtenção de URL

Os recursos de obtenção de URL são todos os recursos envolvidos com o serviço de obtenção de URL do App Engine e não contam com nenhum recurso faturável.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.7: Cotas para recursos do serviço de obtenção de URL.

Recurso	Cota padrão gratuita		Cota com faturamento ativado	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API de obtenção de URL	657.000 chamadas	3.000 chamadas/minuto	46.000.000 chamadas	32.000 chamadas/minuto
Dados enviados para a obtenção de URL	Até o limite da largura de banda de saída	22 megabytes/minuto	Até o limite da largura de banda de saída	740 megabytes/minuto
Dados recebidos da obtenção de URL	Até o limite da largura de banda de entrada	22 megabytes/minuto	Até o limite da largura de banda de entrada	740 megabytes/minuto

3.5.8.1 Chamadas da API de obtenção de URL

Semelhante aos outros serviços, este recurso contabiliza as chamadas à API do serviço de obtenção de URL para realizar solicitações HTTP ou HTTPS.

3.5.8.2 Dados enviados para a obtenção de URL

Representa a quantidade de dados enviados para o serviço de obtenção de URL em solicitações HTTP ou HTTPS. Esta quantidade de dados também é contabilizada para a cota de Largura de banda de saída.

3.5.8.3 Dados recebidos da obtenção de URL

Representa a quantidade de dados recebidos do serviço de obtenção de URL como resposta a solicitações HTTP ou HTTPS. Esta quantidade de dados também é contabilizada para a cota de Largura de banda de entrada.

3.5.9 Manipulação de imagens

Recurso contabilizado sempre que o aplicativo faz uma chamada à API do serviço de manipulação de imagens do App Engine e não contam com nenhum recurso faturável.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.8: Cotas para recursos do serviço de manipulação de imagens.

Recurso	**Cota padrão gratuita**		**Cota com faturamento ativado**	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API de manipulação de imagens	864.000 chamadas	4.800 chamadas/minuto	45.000.000 chamadas	31.000 chamadas/minuto
Dados enviados à API	1 gigabytes	5 megabytes/minuto	560 gigabytes	400 megabytes/minuto
Dados recebidos da API	5 gigabytes	28 megabytes/minuto	427 gigabytes	300 megabytes/minuto
Transformações executadas	2.500.000 transformações	14.000 transformações/minuto	47.000.000 transformações	32.000 transformações/minuto

3.5.9.1 Chamadas da API de manipulação de imagens

O Google App Engine contabiliza todas as chamadas que o aplicativo faz a API do serviço de manipulação de imagens para requisitar acesso a ele.

3.5.9.2 Dados enviados à API

Quantidade total de dados que o aplicativo envia para as funções da API do serviço de manipulação de imagens.

3.5.9.3 Dados recebidos da API

Quantidade total de dados que o aplicativo recebe das funções da API do serviço de manipulação de imagens.

3.5.9.4 Transformações executadas

Mostra o número de vezes que o serviço de manipulação de imagens realizou transformações em dados de imagens pela solicitação do aplicativo.

3.5.10 Cache de memória

Aqui são contabilizados os recursos do serviço de cache de memória sempre que o aplicativo requisita alguns de seus recursos. Este serviço não conta com nenhum recurso faturável.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.9: Cotas para recursos do serviço de cache de memória.

Recurso	**Cota padrão gratuita**		**Cota com faturamento ativado**	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API do cache de memória	8.600.000	48.000 chamadas/minuto	96.000.000	108.000 chamadas/minuto
Dados enviados à API	10 gigabytes	56 megabytes/minuto	60 gigabytes	128 megabytes/minuto
Dados recebidos da API	50 gigabytes	284 megabytes/minuto	315 gigabytes	640 megabytes/minuto

3.5.10.1 Chamadas da API do cache de memória

Recurso que contabiliza o total de vezes que o aplicativo acessou a API de cache de memória para adquirir, definir ou expirar valores.

3.5.10.2 Dados enviados à API

Quantidade de dados que o aplicativo envia ao serviço de cache de memória através da sua API.

3.5.10.3 Dados recebidos da API

Quantidade de dados que o aplicativo recebe do serviço de cache de memória em chamadas à sua API.

3.5.11XMPP

Assim como os outros serviços do App Engine, o serviço XMPP também tem seus recursos monitorados e contabilizados, porém não conta com nenhum recurso faturável.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.10: Cotas para recursos do serviço XMPP.

Recurso	**Cota padrão gratuita**		**Cota com faturamento ativado**	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API XMPP	657.000 chamadas	3.000 chamadas/minuto	46.000.000 chamadas	32.000 chamadas/minuto
Dados enviados pelo XMPP	4 gigabytes	22 megabytes/minuto	1,046 gigabytes	740 megabytes/minuto
Destinatários XMPP	657.000 destinatários	*n/a*	46.000.000 destinatários	*n/a*
Convites XMPP enviados	1.000 convites	*n/a*	100.000 convites	*n/a*

É importante ressaltar que um convite de bate-papo contabiliza para todos os recursos listados acima.

Toda a computação executada pelo tratador das requisições XMPP conta para a cota de tempo de CPU, com um tempo máximo de 50 milissegundos por mensagem, além de contar para os recursos de solicitações e largura de banda de entrada.

3.5.11.1 Chamadas da API XMPP

Contabiliza o total de vezes que a aplicação acessou funções do serviço de mensagens XMPP.

3.5.11.2 Dados enviados pelo XMPP

Representa o total de dados enviados através do serviço XMPP. Esta quantidade de dados também é contabilizada para a cota de Largura de banda de saída.

3.5.11.3 Destinatários XMPP

Número total de destinatários a quem o aplicativo enviou mensagens através do serviço XMPP.

3.5.11.4 Convites XMPP enviados

Número total de convites para bate-papo enviados pelo aplicativo através do serviço XMPP.

3.5.12Task Queue

Contabiliza taxas relativas ao uso do serviço Task Queue do App Engine. Este serviço não possui nenhum recurso faturável e por se tratar de um serviço ainda em desenvolvimento e disponibilizado em caráter experimental os limites podem sofrer mudanças.

A tabela abaixo mostra os limites diários e por minuto para situações com e sem o faturamento ativado:

Tabela 3.11: Cotas para recursos do serviço Task Queue.

Recurso	**Cota padrão gratuita**		**Cota com faturamento ativado**	
	Limite diário	**Taxa máxima**	**Limite diário**	**Taxa máxima**
Chamadas da API Task Queue	100.000	*n/a*	1.000.000	*n/a*

A execução de uma tarefa deste serviço também conta para as cotas de solicitações, tempo de CPU, largura de banda de saída e largura de banda de entrada.

3.5.12.1 Chamadas da API Task Queue

Conta o número de vezes que a aplicação acessou o serviço Task Queue para enfileirar uma tarefa.

3.5.13Blobstore

Contabiliza taxas relativas ao uso do serviço Blobstore do App Engine. Este serviço está disponível apenas para quem possui o faturamento ativado. Os dados armazenados por este serviço contam também para a cota de dados armazenados do serviço de armazenamento de dados. Por se tratar de um serviço ainda em desenvolvimento e disponibilizado em caráter experimental os limites podem sofrer mudanças.

As tabelas abaixo mostram os limites diários e por minuto para situações com e sem o faturamento ativado para os recursos de armazenamento de dados:

Tabela 3.12: Cotas para recursos do serviço Blobstore.

Recurso	Cota padrão gratuita	Cota com faturamento ativado
Dados armazenados do Blobstore	1 gigabyte	1 gigabyte gratuito; sem máximo

Tabela 3.13: Cotas para recursos do serviço Blobstore.

Recurso	Cota com faturamento ativado	
	Limite diário	Taxa máxima
Chamadas da API Blobstore	140.000.000 de chamadas	72.000 chamadas/minuto

Além das cotas específicas, este serviço consome o recurso de tempo de CPU para realizar as suas operações.

3.5.13.1 Chamadas da API Blobstore

Número total de vezes que o aplicativo chamou a API do serviço Blobstore, como por exemplo, para criar URLs de envio de arquivos.

3.5.13.2 Dados armazenados do Blobstore

Total de dados armazenados pelo serviço Blobstore. Também conta para a cota de dados armazenados do sistema de armazenamento de dados.

3.5.14Implementações

Representa o número de vezes que um aplicativo foi enviado pelos seus desenvolvedores. O limite diário para esta cota é de 1000 envios.

3.5.15 Faturamento

Atualmente o Google App Engine possibilita que sejam adquiridas cotas extras para os recursos de largura de banda de saída, largura de banda de entrada, tempo de CPU, dados armazenados e destinatários de e-mail do serviço de mensagens. A tabela abaixo mostra estes recursos juntamente com a unidade básica de faturamento e o respectivo valor da unidade extra:

Tabela 3.14: Recursos faturáveis.

Recurso	Unidade	Custo unitário
Largura de banda de saída	Gigabytes	$0,12
Largura de banda de entrada	Gigabytes	$0,10
Tempo de CPU	Horas da CPU	$0,10
Dados armazenados	Gigabytes por mês	$0,15
Destinatários de e-mail	Destinatários	$0,0001

O Google App Engine cobra os recursos extras em dólares americanos. A descrição de todo o processo necessário para ativar o faturamento é descrito adiante, na seção que aborda o console de administração da aplicação.

3.6 Console de administração

O Console da administração do Google App Engine fornece acesso completo às configurações e às informações da aplicação.

O acesso ao console de administração é feito através do seguinte endereço: *http://appengine.google.com/*.

Após acessar o endereço citado acima serão requisitados dados correspondentes a uma conta Google cadastrada. Caso o App Engine seja usado integrado com o Google Apps o endereço para acessar o console de administração sofre algumas modificações, se tornando o seguinte, onde *seu-dominio.com* é o domínio no Google Apps: http://appengine.google.com/a/*seu-dominio.com*.

Pelo uso do console de administração é possível criar um novo aplicativo, inspecionar dados de acesso, logs de erro e analisar tráfego, dentre outras funcionalidades.

Ao entrar no endereço do console de administração e obter acesso através da autenticação dos dados da conta Google é possível visualizar uma lista dos aplicativos existentes para a conta, de um máximo de 10 aplicativos, contendo o identificador, título e versão atualmente sendo disponibilizada do aplicativo. Caso o limite de 10 aplicativos ainda tenha sido atingido para a conta, há a possibilidade de criação de mais aplicativos. Ao se clicar no identificador do aplicativo é possível ter acesso ao console de administração do aplicativo, o qual está dividido em quatro macro-seções.

3.6.1 Criando uma aplicação

É possível criar uma aplicação clicando no botão *"Create an Application"* acessível através da visualização da lista de aplicativos caso o limite máximo de aplicações por conta não tenha sido atingido para a conta em questão.

Após clicar no botão é possível visualizar um formulário onde devem ser preenchidos o identificador da aplicação, título e especificar as opções de autenticação, onde se deve informar se aplicação é aberta ou restrita a um domínio do Google Apps.

Ao finalizar o processo de criação não é mais possível modificar o seu identificador, o título escolhido será mostrado na página de autenticação por contas Google, caso o aplicativo use o serviço de contas do Google este fim.

3.6.2 Main

A seção *Main* é a seção principal de entrada do console de administração e que engloba os aspectos gerais do aplicativo.

3.6.2.1 Dashboard

O item *Dashboard* desta seção expõe números referentes ao uso dos recursos faturáveis feitos pelo aplicativo além de gráficos representando o uso de recursos por tempo para vários recursos.

Outro painel presente neste item mostra o carregamento atual do sistema. Este painel mostra de modo geral o número de solicitações para cada URI solicitado do aplicativo, média de consumo da CPU em megaciclos na ultima hora e a porcentagem de CPU consumida pelo URI em relação às outras URIs. Estas medidas obedecem à regra de renovação de recursos da cotas.

Semelhante ao painel anterior existe um painel que registra erros na tentativa de carregar um URI, mostrando o total de erros e a porcentagem de erros em relação às requisições deste URI.

3.6.2.2 Quota Details

Este item mostra detalhes de todas as cotas da aplicação mostrando-as em porcentagem e números absolutos de uso na atual diária, agrupadas da seguinte forma:

a) Requisições
b) Armazenamento de dados
c) Mensagens
d) Obtenção de URL
e) Manipulação de imagens

f) Cache de memória
g) XMPP
h) Task queue
i) Implementações

3.6.2.3 Logs

Nos *Logs* é possível ter acesso aos registros gerados pelo aplicativo. Primeiramente será possível visualizar os vinte eventos mais recentes do registro. É possível filtrar os resultados por nível de gravidade, data de registro ou por uma expressão regular.

No rótulo do *log* é possível verificar a data e hora de criação, URL gerador, status retornado, latência e o consumo de CPU.

3.6.2.4 Cron Jobs

Possibilita visualizar todas as tarefas programadas ativas criadas pela aplicação. São apresentadas a URL executada pela tarefa juntamente com sua descrição, programação de execução, data e horário da última execução e último status de execução. As tarefas programadas são definidas na implementação do aplicativo através de arquivos específicos.

3.6.2.5 Task Queue

O item *Task Queue* apresenta detalhes referentes às cotas deste serviço que o aplicativo tenha usado, caso o aplicativo já tenha feito uso do serviço.

Além dos detalhes citados é possível notar um painel informativo contendo o nome das filas existentes, taxa de execução máxima, capacidade, data e hora da última tarefa adicionada, número de tarefas na fila e o número de tarefas executadas no último minuto.

3.6.2.6 Blacklist

O último item mostra uma espécie de lista negra para o aplicativo definida na implementação do aplicativo através da configuração de um arquivo específico.

Este item conta com uma lista de endereços de sub-rede associados com endereços *IP* e uma descrição. Também está disponível uma lista dos 25 *IPs* que mais acessaram a aplicação.

3.6.3 Data

Esta seção agrupa os itens do console de administração relacionados com o sistema de armazenamento de dados do Google App Engine. A partir dela é possível verificar índices, entidades, estatísticas ou visualizar *Blobs.*

3.6.3.1 Datastore Indexes

Lista todos os índices da aplicação mostrando tanto os índices criados automaticamente por ela quanto os criados pelo desenvolvedor através do arquivo de configuração de índices. Esta lista inclui a entidade em que o índice é aplicado, nome do índice e seu status.

3.6.3.2 Datastore Viewer

Através deste item é possível ao administrador visualizar o identificador e propriedades de todas as entidades mantidas pela sua aplicação no armazenamento de dados do App Engine. A escolha das entidades visualizadas pode ser feita tanto por uma caixa de seleção dos tipos existentes, quanto pelo uso de *query* usando a linguagem *GQL*.

Ao clicar no identificador da entidade, a mesma é mostrada em mais detalhes e alterar os valores das suas propriedades.

3.6.3.3 Datastore Statistics

Este é o item que mostra dados estatísticos sobre as entidades armazenadas pelo aplicativo no sistema de armazenamento de dados do App Engine.

A visualização padrão mostra dados gerais sobre todas as entidades, divulgando a ultima atualização de uma entidade, número total de entidades e o tamanho de todas as entidades que estão armazenadas. Este item conta com um painel contendo todos os tipos de propriedades armazenados com o respectivo espaço ocupado, além de dois interessantes e intuitivos gráficos. Um transcreve os tipos de propriedades na porcentagem de espaço utilizado pelo tipo em relação ao total de espaço utilizado por todos os tipos presentes. O outro mostra a porcentagem de espaço utilizado por cada tipo de entidade em relação ao espaço total ocupado.

Ao mudar na caixa de seleção o tipo de estatísticas a mostrar, de todas as entidades, para uma entidade específica, há uma pequena mudança dos dados apresentados. São informados o número total de entidades do tipo em questão e o tamanho médio das entidades do tipo. O painel mostra todas as propriedades destas entidades juntamente com o seu tipo e espaço utilizado em bytes e porcentagem que representa no tamanho total destas entidades. Continua existindo um gráfico de porcentagem de espaço utilizado por cada tipo de propriedade, mas que agora está em relação ao espaço que as propriedades do tipo escolhido ocupam. O outro gráfico mostra o espaço utilizado pelas entidades do tipo em relação ao total de espaço ocupado em porcentagem.

3.6.3.4 Blob Viewer

Este item dá a capacidade de visualizar os *Blobs* criados pelo serviço Blobstore. São mostrados o nome do arquivo, tipo do conteúdo, tamanho e data de criação do *Blob*. É possível filtrar os objetos a serem visualizados por meio de uma caixa de seleção.

3.6.4 Administration

Esta seção é a responsável por cuidar dos aspectos administrativos da aplicação hospedada pelo Google App Engine, podendo verificar e alterar importantes configurações do mesmo.

3.6.4.1 Application Settings

Este item está dividido em configurações básicas, serviços configurados, configurações de domínio e desabilitar ou excluir aplicativo.

A configuração básica torna possível a mudança do título do aplicativo e do tempo para que os cookies expirem, além de mostrar o identificador da aplicação e o método de autenticação escolhido (Contas do Google ou restrito a um domínio do Google Apps).

Outra área deste item mostra os serviços que estão configurados para o uso do aplicativo. Estes serviços precisam ser habilitados explicitamente pelo desenvolvedor, como por exemplo, o serviço XMPP.

Caso o desenvolvedor possua uma conta no Google Apps, é possível associar o aplicativo aos domínios sob posse da conta. Para tanto será necessário se autenticar utilizando uma conta do Google Apps.

Por fim há uma opção capaz de desabilitar a aplicação. Assim que a aplicação é desabilitada o App Engine para de servir requisições para esta aplicação, mas mantém todos os dados e estados inalterados garantindo assim uma futura reabilitação. Com a aplicação desabilitada torna-se possível excluí-la permanentemente. Ao se requisitar a remoção permanente da aplicação é preciso desativar todos os recursos de faturamento, só então é enviado um e-mail informativo para o administrador e inicia-se uma contagem de 72 horas, dentro das quais é possível voltar atrás na decisão. Passado este tempo a aplicação é excluída de forma permanente, sem condições de reversão, liberando espaço para a criação

de outra aplicação, já que há um limite de 10 aplicativos por conta. O nome do aplicativo excluído fica reservado para uso futuro.

3.6.4.2 Permissions

As permissões possibilitam aos administradores do aplicativo convidar outros usuários do sistema de contas do Google para administrarem a aplicação.

Um administrador possui controle total sobre a aplicação, podendo enviar e disponibilizar uma implementação, ter acesso total ao console de administração e inclusive convidar ou excluir administradores.

3.6.4.3 Versions

É possível por meio do console de aplicação ter uma maneira de gerenciar as versões disponíveis do aplicativo. Uma versão é definida no envio da aplicação para a nuvem do Google e deve ser um número inteiro maior que zero.

Este item lista todas as versões existentes do aplicativo, exibindo a data de envio, quem enviou e disponibilizando um link para acesso. É possível excluir uma versão, desde que ela não seja a única disponível. Também há a possibilidade de trocar a versão padrão da aplicação.

3.6.4.4 Admin Logs

O App Engine mantém um registro de todas as ações que os administradores da aplicação realizam usando o console de administração ou a SDK de desenvolvimento, como desabilitar a aplicação ou enviar uma nova versão.

A visualização dos registros pode ser feita listando todos os eventos administrativos ou filtrando-os por tipos específicos.

3.6.5 Billing

Para obter recursos além das cotas gratuitas é necessário ativar o faturamento para o aplicativo. A configuração do faturamento é feita através do item "*Billing Settings*" contido nesta seção, onde também é possível verificar o administrador de faturamento e a alocação de recursos atual.

3.6.5.1 Configurando um orçamento diário

O orçamento diário serve para controlar a quantidade extra de recursos, fazendo com que os gastos não excedam o orçamento. Cada recurso faturável possui seu orçamento individual, não havendo transferência automática dos valores dos recursos caso um deles acabar.

Para definir um orçamento diário é preciso habilitar o faturamento através de um botão disponível na página. Após habilitar o faturamento, é preciso inserir o orçamento diário máximo em dólares americanos, representando o máximo que se está disposto a pagar em um período de 24 horas. É importante ressaltar que não necessariamente este valor será sempre cobrado, pois a cobrança é feita apenas por aquilo que é usado, sendo estes, os valores máximos que podem ser cobrados. O número de centavos cobrados deve ser inteiro com unidades mínimas de um centavo, caso o uso de um recurso obtenha valores fracionados entre um centavo e outro, este valor será arredondado para cima.

Por padrão o orçamento diário máximo é automaticamente alocado dentre os recursos faturáveis, porém é possível selecionar outro orçamento predefinido ou personalizar a alocação do orçamento dentre os recursos manualmente.

Estando definida a distribuição do orçamento diário máximo, deve-se selecionar o país do dono da aplicação e clicar no botão "*Google Checkout*", que redirecionará para a página que solicita as informações de pagamento, o qual deve ser feito através de um cartão de crédito internacional. A cobrança é feita semanalmente, e os valores são em dólares americanos.

Finalizados os passos de habilitação do faturamento, o status de faturamento do aplicativo indica "*Ativando o faturamento*" por até 20 minutos, não sendo permitida nenhuma alteração no orçamento durante este período. O status mudará para "*Faturamento ativado*" caso o Google App Engine receba a confirmação do *Google Checkout*.

Com o sucesso da ativação do faturamento para a aplicação há uma mudança nas informações da página principal do console de administração, ajustadas para informar sobre o orçamento definido.

3.6.5.2 Alterando o orçamento diário

Depois de ativado o faturamento, o orçamento máximo diário pode ser aumentado a qualquer momento, sendo alterado do mesmo modo quando foi feita a ativação, porém será necessário acessar o *Google Checkout* para autorizar o novo limite de cobrança semanal. Qualquer aumento entrará em vigor de entre 15 e 30 minutos após a confirmação da autorização de pagamento.

A diminuição do orçamento máximo diário, por outro lado, não exige uma nova autorização de cobrança do *Google Checkout*. As reduções entram em vigor cerca de 15 minutos após a alteração, entretanto em casos do uso de algum recurso estar muito próximo do seu novo limite ou o tiver ultrapassado será necessário esperar a renovação do recurso para aplicar as novas cotas.

Qualquer administrador que não seja o administrador de faturamento pode redistribuir o orçamento entre os recursos, desde que a nova distribuição não exceda o orçamento diário máximo previamente definido. Para elevá-lo o administrador precisa assumir a responsabilidade sobre o faturamento e autorizar um novo limite de cobrança semanal.

3.6.5.3 Administrador de faturamento

O administrador de faturamento é o administrador da aplicação responsável por ajustar o orçamento diário e pelo seu pagamento semanal. Qualquer administrador pode administrar o faturamento clicando em "*Take over billing*" e tendo seus dados aceitos para a cobrança.

3.6.5.4 Billing History

O App Engine registra todos os eventos relacionados ao faturamento do aplicativo e os disponibiliza para visualização através deste item. O App Engine também gera relatórios diários de uso de recursos e cobranças.

3.7 Conclusão

O desenvolvimento e aprimoramento por parte de Google das suas tecnologias de armazenamento de dados e de computação distribuída propiciou a sua entrada no ramo da computação em nuvem. Isto se deu em um primeiro momento através do fornecimento de software como um serviço, para posteriormente, com o amadurecimento dos conceitos relacionados a este paradigma, desenvolver sua própria plataforma como serviço, possibilitando um uso mais amplo da sua infraestrutura.

Este serviço, muito por conta de ainda não estar em um estágio maduro o suficiente, apresenta algumas objeções, principalmente no que diz respeito aos limites impostos, falta de ferramental adequado para lidar com o sistema de armazenamento de dados ou sistema de cotas não muito claro em alguns aspectos, apesar de um bom sistema de administração.

Contudo é importante ressaltar que este é o começo de um novo paradigma para o modo de servir e disponibilizar aplicativos web, abrindo nichos antes não explorados pela computação, mas agora possíveis com a maior facilidade de se obter recursos computacionais por demanda e escaláveis.

4. Google Application Engine e Java

4.1 Introdução

O Google Application Engine disponibiliza seu ambiente de execução, servidor de desenvolvimento e APIs de serviços tanto para a linguagem Python quanto para a linguagem Java. Devido à sua popularidade e às facilidades de desenvolvimento, graças à completa integração com a IDE de desenvolvimento Eclipse, será abordada neste trabalho a integração entre o App Engine e a linguagem Java.

O ambiente Java do App Engine executa em uma JVM (Máquina Virtual Java) da versão 6 do Java, com suporte a servlets, biblioteca Java padrão, armazenamento de dados e serviço do App Engine. Apesar de executar os programas usando a versão 6 do Java, pode-se usar classes compiladas utilizando qualquer versão anterior. O suporte as bibliotecas padrões facilita o desenvolvimento de aplicações já que não requer grandes mudanças no desenvolvimento para o App Engine em relação aos servidores usuais. O App Engine também possui total integração com a IDE Eclipse através de um plugin contendo sua SDK (Kit de desenvolvimento local) completa e também do Google Web Toolkit. A SDK do Google App Engine possui suporte para a versão 5 do Java e posteriores, além de poder ser usado com os vários *frameworks* de desenvolvimento Java para web, como o *Struts* ou *Spring*.

O aplicativo é estruturado na forma padrão WAR (*Web Application Archive*) – semelhante ao padrão JAR do Java para aplicações desktop – utilizado para desenvolver aplicativos baseados na versão Java para servidores web conhecida como *Java 2 Enterprise Edition* ou *J2EE*. Há a separação do código e dos arquivos estáticos, além de um descritor de implementação na forma de um arquivo xml com nome *web.xml* juntamente com outros arquivos de configuração.

O armazenamento de dados do App Engine para Java suporta *JDO* (*Java Data Objects*) ou *JPA* (*Java Persistence API*), padrões Java para armazenamento de dados. Estes padrões para o armazenamento de dados são implementados pelo App Engine com o uso do DataNucleus Access Platform, a implementação de software livre escolhida pelo App Engine para dar suporte a estes padrões.

Todos os serviços disponibilizados pelo App Engine possuem sua implementação para Java, disponíveis através de APIs específicas. Além destes serviços o kit de desenvolvimento possui ferramentas para testes, um servidor local de desenvolvimento capaz de simular o ambiente real de execução do App Engine, integração com Google Web Toolkit e uma ferramenta capaz de realizar certas configurações no aplicativo chamada de *AppCfg.*

4.2 O ambiente Java

O Google App Engine detecta e ativa automaticamente o ambiente Java para executar a aplicação no momento em que ela é enviada à nuvem do Google através do plugin para Eclipse ou pelo *AppCfg.* Uma mesma aplicação pode ter versões em Python e em Java coexistindo tranquilamente, sendo a versão ativa o determinante de qual ambiente utilizar.

O ambiente Java é suportado pela tecnologia Java Enterprise Edition que fornece recursos para implementar softwares distribuídos executados em servidores específicos e classes específicas destinadas a servirem às solicitações web, conhecidas como *Servlets*. As classes *Servlets* basicamente possibilitam que as classes que atendem solicitações específicas as herdem e implementem os métodos capazes de receber os dados e enviar a resposta.

4.3 Sandbox

Como foi enunciado no capítulo anterior, o Google Application Engine restringe o aplicativo ao seu respectivo ambiente isolado, denominado sandbox. A sandbox tem a finalidade de garantir que um aplicativo não interfira na execução de outro, além de, por se tratar de um ambiente distribuído, servir como uma forma de virtualização de um sistema operacional.

A sandbox do aplicativo implementa restrições quanto ao acesso ao sistema de arquivos, impedindo o uso de classes que façam escrita de arquivos e restringindo a leitura aos arquivos estáticos enviados juntamente com o aplicativo.

Outras restrições passíveis de nota dizem respeito à proibição imposta a aplicação em criar subprocessos paralelos ao processo que executa a solicitação corrente, a fim de evitar sobrecarga nos servidores, proibição do uso de alguns recursos da classe *java.lang.System* e o uso de reflexão em classes que não sejam as pertencentes ao aplicativo.

4.4 Limites

Além das cotas descritas anteriormente, os *Servlets* responsáveis por tratar cada solicitação estão sujeitos às seguintes limitações:

a) Máximo de 10 megabytes tanto para o tamanho da solicitação, quanto para o tamanho da resposta.
b) Uma solicitação não pode demorar mais do que 30 segundos para retornar a resposta.
c) Máximo de 1000 arquivos de código.
d) Máximo de 1000 arquivos estáticos, como imagens, folhas de estilo, arquivos de configurações, etc.
e) Um arquivo, tanto de código fonte, quanto estático não pode ter mais do que 10 megabytes.
f) O tamanho total de todos os arquivos somados do aplicativo não pode ultrapassar 150 megabytes.

4.5 Servidor de desenvolvimento

O Google App Engine disponibiliza para a criação dos aplicativos um servidor de desenvolvimento local que visa simular da forma mais fiel possível o ambiente real de execução do aplicativo provendo simulação do sistema de armazenamento de dados, serviços e restrições, além de suporte à depuração e testes.

Para a simulação do armazenamento de dados está embutida no servidor de desenvolvimento a implementação dos padrões Java para armazenamento utilizados pelo App Engine, conhecida como DataNucleus. As entidadades e índices são armazenados sob a forma de um arquivo com nome *local_db.bin* criado no diretório *WAR* do aplicativo.

Por conta da popularidade da IDE Eclipse para a criação de aplicações Java e do encorajamento de seu uso pelo Google através de sua fácil e total integração por meio de um plugin específico, tornando o processo de desenvolvimento do aplicativo mais leve e prático, será dada ênfase a esta IDE no desenvolvimento de aplicativos para o App Engine.

Antes de começar a usar o servidor de desenvolvimento do App Engine deve-se ter a versão 5 ou 6 da máquina virtual Java, juntamente com suas bibliotecas padrões instaladas na máquina de desenvolvimento. Para isso deve-se acessar o site oficial do Java, realizar o download do *JDK* (Java Development Kit) do *Java2EE* ou *Java SE* e instalá-lo adequadamente

4.6 Plug-in para Eclipse

Visando tornar ainda mais fácil a experiência de desenvolver aplicativos para o App Engine, a equipe do Google desenvolveu um plug-in especialmente para a

IDE de desenvolvimento Eclipse, a qual oferece um abrangente suporte a extensões.

Este plug-in atende a todas as necessidades de desenvolvimento local, incluindo por completo o servidor de desenvolvimento além da SDK do Google Application Engine e SDK do Google Web Toolkit.

4.6.1 Instalação

A instalação se dá de forma prática, através do seguinte procedimento:

a) Obter através do seu site oficial o software Eclipse para desenvolvedores Java EE, que até o presente momento se encontra na sua versão *Galileo 3.5*.
b) Selecionar o menu "*Help > Install New Software...*". A seleção deste menu resultará na abertura de uma janela para a seleção de novos componentes a serem instalados.
c) Nesta janela deverá ser clicado o botão "*Add...*", e na nova janela adicionar o site *http://dl.google.com/eclipse/plugin/3.5*. Note que o site referido pode mudar de acordo com a versão do Eclipse.
d) A adição do site acima resultará no aparecimento das opções *Plugin* – contendo a integração do App Engine com o Eclipse – e SDKs – possui tudo que necessário para o desenvolvimento local, incluindo bibliotecas, servidor de desenvolvimento e kit de desenvolvimento para o Google Web Toolkit –, que deverão ser selecionadas para completa integração.
e) Após a seleção dos componentes deve-se clicar no botão "*Next*" e seguir os passos para a instalação.
f) Após o processo de instalação ser concluído, confirme a solicitação para reiniciar o Eclipse.

4.6.2 Criar um projeto

A criação de um projeto para o Google App Engine utilizando a IDE Eclipse pode ser feita acessando o menu "*File > New > Web Application Project"* ou através de um botão específico, ilustrado na figura abaixo.

Figura 4.1: O quadro mostra o botão para criar um novo projeto.

A ação acima resultará em uma janela para preenchimento dos dados do projeto e seleção da SDK utilizada. Após todas as informações serem preenchidas, o plug-in trata de criar toda a estrutura de diretórios do aplicativo, tornando apto o seu desenvolvimento.

4.6.3 Executar um projeto

Para executar um projeto deve-se acessar o menu "*Run > Run As > Web Application*". A partir daí o servidor será carregado, o projeto processado, e no caso de sucesso, será mostrada uma mensagem no console do Eclipse, juntamente com o endereço para acessar o aplicativo, o qual por padrão é: *http://localhost:8080/.* De posse deste endereço basta utilizá-lo como URL no navegador de preferência para executar o aplicativo.

Além da execução simples do projeto, é possível executá-lo com auxílio de um depurador integrado com o próprio Eclipse acessado através do menu "*Run > Debug As > Web Application*".

Há ainda a possibilidade de alterar as configurações de execução, como por exemplo a porta em que o aplicativo estará disponível, para isso deve-se acessar o menu “*Run > Run Configurations...*”

Uma vez iniciado o servidor com o projeto em execução não se faz necessário interromper e iniciar novamente o servidor para fazer valer qualquer alteração no código fonte, arquivos estáticos, *JSPs (Java Server Pages)*, etc., com exceção de alterações feitas no descritor *web.xml*, onde este procedimento é necessário.

4.6.4 Enviar o aplicativo para o Google Application Engine

O envio do aplicativo para a nuvem de servidores do App Engine é um procedimento simples, iniciado através de um botão contendo o logotipo do App Engine, disponível na barra de ferramentas do Eclipse.

Figura 4.2: Botão para enviar o aplicativo ao Google App Engine.

Na janela que sucede a operação anterior é necessário informar o projeto a ser enviado, e-mail de administrador da aplicação, senha da conta referente ao e-mail informado e, através do clique no texto “App Engine project settings”, o identificador de uma aplicação existente, ou seja, já criada a partir do console de administração, e o número da versão a ser enviada. É importante ressaltar que as informações do último procedimento podem ser modificadas através da alteração do arquivo de configuração *appengine-web.xml*.

Todo o desenvolvimento do envio do aplicativo pode ser acompanhado através do console do Eclipse.

4.7 Bibliotecas adicionais

O App Engine possibilita que o aplicativo utilize bibliotecas de terceiros podendo fazer completo uso desde que estas não infrinjam as restrições impostas pelo App Engine, como por exemplo, a criação de subprocessos ou conexões soquetes diretas.

Isto possibilita que a aplicação utilize frameworks de grande difusão para o desenvolvimento web utilizando Java.

4.8 Estrutura de diretórios

O projeto do aplicativo é estruturado em dois macros diretórios. Um para arquivos de código e outro para arquivos estáticos e de configuração.

O diretório *src/* é constituído pelos arquivos de código do projeto. Nele se encontram os pacotes e arquivos Java. Apesar de este não ser o diretório dos arquivos de configuração, nele se encontra o arquivo de configuração do padrão do banco de dados utilizado.

No diretório *war/* situam-se os demais arquivos. Diretamente neste diretório encontram-se diretamente os arquivos estáticos, como arquivos de imagens e arquivos HTML, JSP e CSS. Há ainda neste diretório outro diretório, denominado *WEB-INF/*, nele se encontram os arquivos de configuração, bibliotecas e o arquivo gerado pelo servidor de desenvolvimento para simular o armazenamento de dados.

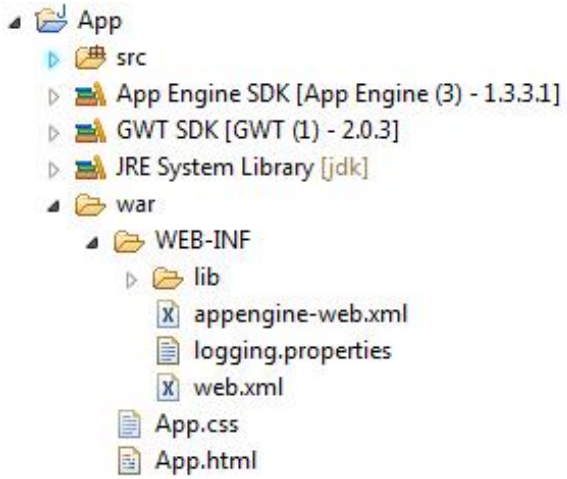

Figura 4.3: Estrutura de diretórios do projeto.

4.9 Configuração

Este item engloba os aspectos acerca da configuração do aplicativo em Java, explorando o descritor de implementação necessário para os aplicativos web Java em geral, arquivo de configuração específico para aplicativos do App Engine, configuração de índices e tarefas programadas.

4.9.1 Descritor de implementação

O descritor de implementação nada mais é que o arquivo *web.xml* encontrado no diretório *war/WEB-INF/* e que trata dos aspectos gerais da configuração de um aplicativo Java para web, fazendo parte do padrão Java para aplicativos web.

Trata-se de um arquivo XML que possui informações de configuração do aplicativo a serem usadas pelo servidor. Por se tratar de um arquivo XML, as etiquetas dos elementos são diferenciáveis em maiúsculas e minúsculas, devendo neste caso serem usadas com letras minúsculas. O nó principal é o elemento <*web-app*>, dentro do qual se pode incluir uma série de outros elementos para configuração específica. Neste item serão abordas apenas as configurações mais

relevantes para este contexto, entretanto é possível saber mais sobre o assunto através da documentação do padrão *Java Servlet*.

4.9.1.1 Mapeamento de servlets

Sua utilidade mais empregada se faz ao mapear um endereço para um servlet capaz de tratar a solicitação. Isto é feito com o uso dos elementos <*servlet*> (define um servlet) e <*servlet-mapping*> (associa um servlet definido como tratador para um endereço). Através do elemento <*servlet*> se define informações do servlet, como nome, classe Java e parâmetros de inicialização. O elemento <*servlet-mapping*> mapeia o servlet a um endereço através do nome especificado anteriormente, ou seja, específica para qual servlet a solicitação será enviada ao acessar o endereço. Um caso prático de uso acontece no caso do serviço XMPP onde as mensagens são enviadas para o endereço */_ah/xmpp/message/chat/*, o qual deve possuir um servlet associado para tratar a solicitação.

```
<servlet>
  <servlet-name>adminControl</servlet-name>
  <servlet-class>voting.server.admin.AdminServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>adminControl</servlet-name>
  <url-pattern>/admin/admin</url-pattern>
</servlet-mapping>
```

Figura 4.4: Trecho de código de mapeamento de servlet.

Os arquivos conhecidos como JSP, que misturam código estático, geralmente HTML, com código Java, são transformados em servlets no processo de compilação e possuem um mapeamento automático utilizando seu caminho completo com o nome do arquivo a partir do diretório raiz. É possível mapeá-lo para outros endereços de acordo com o processo anterior, com a diferença de se especificar o JSP em questão ao invés da classe do servlet.

4.9.1.2 Segurança

O Google App Engine possibilita de forma fácil através de algumas linhas de configuração que o aplicativo se integre ao sistema de contas do Google. Através do elemento <*security-constraint*> é possível especificar recursos, geralmente endereços, que soferão algum tipo de restrição. Neste elemento também se especifica a restrição, onde deve constar se o acesso é permitido para todos os usuários autenticados através do valor "* *(asterisco)*" ou permitir que apenas administradores do aplicativo tenham acesso, por meio do valor "a*dmin"*. Caso haja uma especificação de restrição e as permissões acima citadas não se verifiquem para o recurso especificado, o App Engine trata de redirecionar automaticamente o aplicativo para a pagina de autenticação.

Outro aspecto relacionado à segurança abordado no descritor de implementação diz respeito a endereços seguros através do protocolo *HTTPS*, e se dá também por meio do elemento <*security-constraint*>. Esta configuração restringe o acesso a um recurso apenas por meio de conexões seguras *HTTPS*, negando acesso caso contrário. Todo acesso por conexões seguras deve ser feito por meio do domínio *appspot.com*, não sendo suportado o acesso através do domínio personalizado do Google Apps.

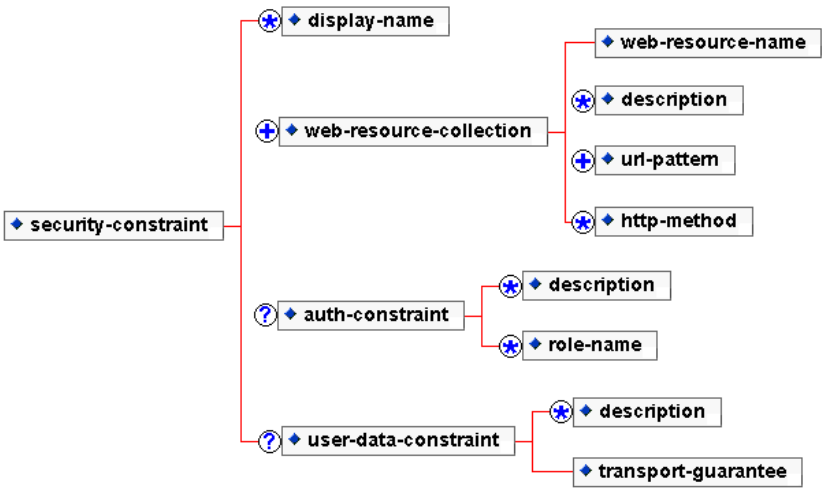

Figure 4.1: Estrutura do elemento security-constraint. Fonte: Documentação Java Servlets.

4.9.1.3 Demais recursos

O padrão Java Servlets especifica uma série de elementos passíveis de serem usados no descritor de implementação do aplicativo. Um destes elementos serve para definir uma lista de arquivos que o servidor deve procurar quando se acessa um subdiretório que não possui um mapeamento explicito antes de retornar o código de erro. A maneira como os erros HTTP ou exceções geradas por servlets são mostrados pelo aplicativo também pode ser modificada associando um recurso a ser exibido no caso do erro se confirmar.

4.9.1.4 Restrições

Os seguintes elementos do descritor de implementação não são suportados, ou são parcialmente:

a) <*security-role*> e <*login-config*> da parte de segurança.
b) <*load-on-startap*> é suportado, entretanto o carregamento ocorre na primeira solicitação feita.
c) <*env-entry*>
d) <*resource-ref*>
e) <*distributable*>
f) <*run-at*>

4.9.2 Configuração do aplicativo

Assim como o descritor de implementação, a configuração do aplicativo também se dá por meio por um arquivo XML no diretório *war/WEB-INF/*. Trata-se do arquivo *appengine-web.xml* e armazena configurações específicas do App Engine.

Este arquivo constitui-se do elemento <*appengine-web-app*> como raiz e deve obrigatoriamente ter os elementos <*application*> e <*version*>, que designam respectivamente o identificador do aplicativo e a versão.

Outra possibilidade é de especificar quais arquivos serão tratados como estáticos e quais serão tratados como recursos. Arquivos estáticos podem ser acessados apenas pelo navegador do usuário, possuindo uma performance maior sobre os arquivos de recursos, os quais podem ser usados pelo código através de métodos de acesso ao sistema de arquivos. Por padrão todos os arquivos são disponibilizados tanto de modo estático como recurso, com exceção dos contidos no diretório *war/WEB-INF/*. As diretrizes dos arquivos que não se encontram no diretório *war/WEB-INF/* podem ser modificadas com o uso do elemento <*static-files*>.

Neste arquivo também é feita a configuração de propriedades do sistema e variáveis de ambiente, ativação do uso de conexões seguras e habilitação do uso de sessões.

4.9.3 Arquivo de índices

O arquivo *datastore-indexes.xml* providencia a configuração dos índices personalizados do aplicativo. Para cada índice deve ser informado o tipo da entidade, uso de filtros por ancestral e as propriedades indexadas. Cada propriedade possui seu nome e a direção de ordenamento no índice. Deve-se tomar cuidado com a quantidade de propriedades de cada índice, ainda mais se forem utilizadas propriedades com vários valores, a fim de evitar a criação de índices explosivos, que aumentam o espaço utilizado de forma exponencial, atingindo os limites rapidamente.

Com a criação e configuração deste arquivo o App Engine deixa de gerar índices automaticamente, a menos que se especifique o contrário neste mesmo arquivo atribuindo o valor “*true*” a propriedade *autoGenerate* do elemento raiz.

4.9.4 Tarefas programadas

As tarefas programadas são definidas com o uso do arquivo de configuração *com.xml*. Deve-se informar a URL do servlet a ser executado, descrição e intervalo de execução para cada tarefa programada.

As tarefas programadas podem executar servlets restritos a administradores e são criadas ou atualizadas juntamente com o envio da aplicação para os servidores do Google.

4.10 AppCfg

O AppCfg é um pequeno aplicativo localizado na pasta do plug-in do App Engine capaz de realizar certas operações de configuração, como limpeza de índices ou o próprio envio do aplicativo.

4.11 Armazenamento de dados

O App Engine para Java procura utilizar padrões para persistência de dados do Java para intermediar a comunicação entre aplicação e o armazenamento de dados. Estes padrões procuram fornecer uma modelagem genérica para os dados, de forma que não seja dependente de um único sistema de armazenamento de dados, dependendo da implementação do padrão mapear o modelo para os sistemas de armazenamento específico.

Os padrões com condições de uso em aplicativos do App Engine são o JPA (*Java Persistence API*) e o JDO (*Java Data Objects*), muito semelhantes entre si. Este último, o JDO, é o mais incentivado pelo Google e o abordado neste trabalho, muito por conta da implementação do seu padrão, conhecida como *DataNucleus*, por se tratar de um software livre de código aberto e também por possuir uma melhor integração com sistemas de armazenamento de dados não relacionais, como o *BigTable*, utilizado pelo App Engine. Framework de vasta difusão, como *HIbernate*

não são suportados, apesar de ter sua base no JPA, por se tratar de um framework específico para bancos de dados relacionais.

O decorrer deste item aborda alguns detalhes acerca do uso do sistema de armazenamento de dados do Google App Engine com Java utilizando o padrão JDO para este propósito.

4.11.1 JDO

A sigla JDO significa *Java Data Objects*, ou algo como objetos de dados Java. Trata-se de um padrão Java para persistência de dados independente do sistema de banco de dados utilizado, ou seja, uma aplicação que utiliza este padrão para modelar suas entidades de dados pode utilizar como sistema de armazenamento o BigTable do Google ou migrar para algum banco de dados relacional, como o *MySQL*, sem que isso afete seu modelo de dados. Deste modo perde-se um pouco em eficiência de modelagem, mas se ganha muito em portabilidade.

Para que isto seja possível é necessário que haja uma implementação do padrão executando em nível de servidor JVM, responsável por mapear os modelos fisicamente ao sistema escolhido. O responsável por esta tarefa no App Engine é o *DataNucleus Access Platform* implementando a versão 2.3 do JDO.

A modelagem das entidades baseia-se nos chamados *POJOs* (*Plain Old Java Objects*), que nada mais são do que objetos simples dotados de atributos ou propriedades, e métodos de acesso (atribuição e obtenção) a elas, adicionados de anotações JDO para torná-los passíveis de persistência.

4.11.2 Anotações JDO

O mapeamento de uma classe para o armazenamento de dados utilizando JDO pode ser feito de três maneiras:

a) Configuração de um arquivo XML especificando todos os detalhes de mapeamento.
b) Uso de anotações Java diretamente nas classes dos modelos.
c) Utilização de API de metadados via código.

Por se tratar do modo mais comum e intuitivo de mapeamento, será utilizado neste trabalho o mapeamento através de anotações Java. Anotações é uma forma de atribuir metadados diretamente no código fonte das classes de modo que os atributos ou métodos anotados sofram algum processamento especial.

O JDO conta com uma série de anotações disponíveis para realizar o mapeamento das classes para o armazenamento de dados. Alguns deles serão descritos adiante ao se mostrar como a modelagem de uma classe é realizada.

```
@Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
private String id;

@Persistent
private String title;
```

Figura 4.5: Exemplo de uso de anotações JDO.

4.11.3 Ciclo de vida dos objetos

Um objeto persistente via JDO pode possuir diferentes estados quando de encontra na memória, utilizados pela implementação JDO para controlá-lo. É importante conhecer esta realidade, pois dependendo do estado em que o objeto se encontra, este possui dados não visíveis na sua modelagem, injetados pelo JDO.

Os objetos JDO podem possuir três macro-estados que merecem destaque:

a) *Transient* ou transitório – Este é o estado em que se encontra um objeto recém criado que ainda não foi persistido e não possui informações adicionais embutidas pelo JDO.
b) *Hollow* ou oco – Representa um objeto persistido, porém sem os valores de suas propriedades carregados.
c) *Persistent* ou persistente – Representa um objeto persistido com os valores das suas propriedades carregados.

É possível destacar (*detach*) um objeto e uma classe definida como destacável da sua conexão com o armazenamento de dados. Quando isto é feito o objeto se torna um objeto persistente, perdendo as informações da conexão através da qual foi obtido, porém armazena uma série de outras informações JDO, como o campo *jdoDetachedState*, que propiciarão ser associado novamente. Um objeto destacado pode ser serializado.

Ao se destacar um objeto de uma classe não destacável, será retornado um objeto transitório, ou seja, sem informações JDO e associação com algum objeto existente, sendo tratado como um objeto novo ao se tentar associá-lo novamente ao armazenamento.

Para ilustrar os estados que um objeto pode apresentar, vamos examinar o caso da inclusão de um objeto. Este objeto começa com um estado transitório, após o uso de funções JDO para armazená-lo ele torna-se persistente, por fim quando o armazenamento efetivamente é realizado ele torna-se um objeto oco, com informações sobre a persistência, porém sem os valores das suas propriedades carregados.

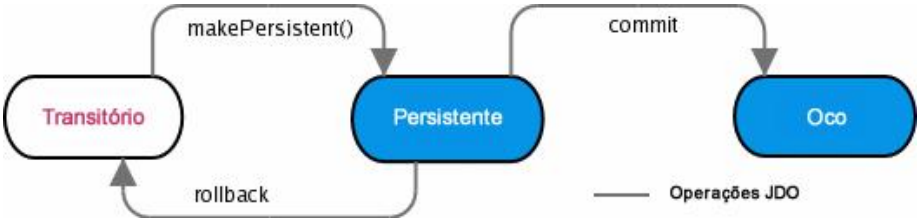

Figura 4.6: Estados apresentados pelo objeto durante uma operação de inclusão. (Traduzido de: http://db.apache.org/jdo/state_transition.html)

4.11.4Modelagem de entidades

Uma entidade capaz de ser armazenada no sistema de armazenamento deve ser modelada com JPO utilizando uma classe simples, contendo atributos, métodos de acesso aos atributos e anotações JDO correspondentes.

O nome da entidade será definido pelo nome da classe, e os nomes das propriedades que serão armazenadas serão os mesmos definidos no modelo. A anotação *@PersistenceCapable* define que a classe é apta a ser persistida. Esta anotação permite uma série de atributos, como o atributo *detachable,* que informa se a classe é destacável.

4.11.4.1 Chave

Toda entidade precisa de um atributo que funcione como identificador único e exclusivo através de uma anotação especial. Este atributo é conhecido como chave da entidade e deve ser capaz de armazenar a chave tal como foi descrita no capítulo anterior.

Há quatro tipos válidos capazes de armazenar a chave de uma entidade. São eles:

a) *Long* – Chaves deste tipo só podem ser preenchidas automaticamente pelo armazenamento de dados e devem ser usados apenas por entidades sem pais.

b) *String* não codificada – Serve para entidades sem pais em que o seu valor é fornecido explicitamente pelo aplciativo.
c) *Key* – Classe específica da biblioteca do App Engine destinada exclusivamente ao armazenamento de chaves completas de entidades que possuam ou não um pai. Possui na sua composição um campo identificador que pode ser atribuído explicitamente pelo aplicativo ou gerado de forma automática pelo armazenamento de dados.
d) *Key* como *String* codificada – Possui todos os benefícios do tipo *Key* com as vantagens de ser representada como uma cadeia de caracteres codificada, garantindo assim uma maior portabilidade.

A API do Google App Engine disponibiliza a classe *KeyFactory* exclusivamente para lidar com chaves do tipo *Key*.

As chaves são definidas através do uso conjunto das anotações *@PrimaryKey* e *@Persistent* juntamente com seus atributos. Para o caso específico de uso da chave *Key* como *String* codificada há a necessidade da seguinte anotação sobre uma propriedade do tipo *String*:

- @Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")

4.11.4.2 Propriedades

As propriedades da entidade devem aparecer na classe de modelo normalmente como atributos da classe adicionados da anotação *@Persistent* para explicitar que a propriedade deve ser armazenada. A especificação JDO determina que alguns tipos de propriedades devem ser persistidos automaticamente, para isto existe a anotação *@NotPersistent* caso o campo não deva ser persistido.

As propriedades persistidas devem possuir nível de encapsulamento *private* ou *protected*, métodos de acesso para atribuição ou leitura de valores individuais.

São suportadas propriedades dos principais tipos de valor, além de coleções e objetos personalizados que implementam a classe *java.io.Serializable*.

Para propriedades que necessitam ser capazes de armazenar múltiplos valores devem usar um tipo de propriedade capaz de armazenar uma coleção. Os seguintes tipos de coleções são suportados:

a) java.util.ArrayList
b) java.util.HashSet
c) java.util.LinkedHashSet
d) java.util.LinkedList
e) java.util.List
f) java.util.Set
g) java.util.SortedSet
h) java.util.Stack
i) java.util.TreeSet
j) java.util.Vector

Objetos contendo dados serializados devem ser armazenados com o uso da propriedade do tipo *Blob*, ou definindo a anotação *@Persistent(serialized = "true")* caso i tipo da propriedade seja uma classe que implementa *java.io.Serializable*. Estes tipos de propriedades não podem ser indexados, resultando que não podem ser usados em consultas.

```
public class Option {
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    @Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
    private String id;

    @Persistent
    private String title;

    // ...
}
```

Figura 4.7: Classe modelada com anotações JDO.

4.11.4.3 Aprimoramento

Antes de serem enviadas para o servidor, as classes de modelo devem passar por um aprimoramento pela implementação do JDO, incluindo código extra necessário para a persistência funcionar corretamente. Quando se usa o plug-in para eclipse este aprimoramento é feito automaticamente no ato de compilação da classe.

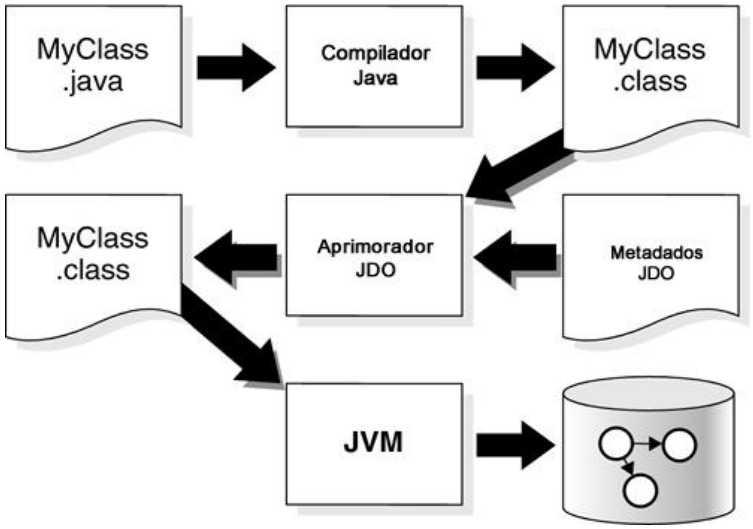

Figura 4.8: Esquema de aprimoramento de classes. (Traduzido de: http://www.informit.com)

4.11.5Relacionamentos

Os relacionamentos entre entidades podem ser de dois tipos: proprietários e não-proprietários. Relacionamentos proprietários definem que uma entidade pai controla o ciclo de vida da outra, como a sua criação e exclusão. Caso uma entidade pai proprietária seja excluída, as entidades filhas também o são em cascata. Por outro lado relacionamentos não proprietários não possuem este controle, ficando isto a cargo do aplicativo.

Quando um relacionamento proprietário é estabelecido, uma entidade se torna pai de outra entidade, fazendo assim, as duas, parte do mesmo grupo de entidades. Uma entidade pode possuir apenas um pai.

Relacionamentos proprietários podem ser do tipo um-para-um ou um-para-vários. Os relacionamentos não proprietários podem ser do tipo um-para-um, um-para-vários ou vários-para-vários.

4.11.5.1 Relacionamento proprietário de um-para-um

Em um relacionamento proprietário de um-para-um unidirecional a classe pai possui como propriedade persistente um objeto da classe filha. Já para um relacionamento deste tipo, mas bidirecional, a classe filha também possui uma propriedade persistente da classe pai, porém marcada com a anotação *@Persistent(mappedBy="propriedadeFilha")*, onde "*propriedadeFilha"* recebe o nome da propriedade filha na classe pai.

Classe Pai:

```
@Persistent
private EntidadeFilha filha;
```

Classe filha:

```
@Persistent(mappedBy="filha")
private EntidadePai pai;
```

Figura 4.9: Relacionamento proprietário de um-para-um bidirecional.

4.11.5.2 Relacionamento proprietário de um-para-vários

O relacionamento proprietário de um-para-vários é semelhante ao de um-para-um descrito anteriormente com a diferença de a classe pai possuir uma coleção de objetos da classe filha. Relacionamentos bidirecionais também ocorrem de forma muito parecida, porém neste caso a propriedade da classe pai recebe *@Persistent(mappedBy="propriedadePai"), onde "propriedadePai"* recebe o nome da propriedade pai na classe filha.

Classe Pai:

```
@Persistent(mappedBy = "pai")
private List<EntidadeFilha> filhas;
```

Classe filha:

```
@Persistent
private EntidadePai pai;
```

Figura 4.10: Relacionamento proprietário de um-para-vários bidirecional.

4.11.5.3 Relacionamento não proprietário de um-pra-um

Ocorre de forma semelhante ao relacionamento proprietário, porém ao invés de possuir um objeto de outra classe como propriedade, deve-se ter como propriedade a chave da entidade a qual se pretende relacionar.

4.11.5.4 Relacionamento não proprietário de um-para-vários

Semelhante ao relacionamento anterior porém ao invés de apenas uma chave como propriedade, deve-se ter uma coleção de chaves que referenciem outras entidades.

4.11.5.5 Relacionamento não proprietário de vários-para-vários

Este tipo de relacionamento ocorre quando ambos os lados do relacionamento possuem uma coleção de chaves de entidades. Por exemplo, a modelagem da entidade *entidade1* possui uma coleção de chaves de entidades do tipo *entidade2*, a modelagem desta por sua vez, também possui uma coleção de chaves de entidades do tipo *entidade1*.

4.11.6 Manipulação de objetos

Todo o acesso ao serviço de armazenamento de dados usando JDO deve ser realizado por intermédio de uma instância da classe *PersistenceManager* obtida

através de uma instância da classe *PersistenceManagerFactory*. Instâncias desta última classe são relativamente demoradas para inicializarem, o que encoraja o uso do padrão de projeto *Singleton*, onde uma única instância da classe serve a aplicação durante toda sua execução.

Uma instância da classe *PersistenceManager* possui uma única transação capaz de ser obtida para uso. Após finalizar as operações a instância deve ser fechada através do método *close*, inviabilizando seu uso posterior. Acessos posteriores ao fechamento de uma instância devem ser realizados obtendo-se outra instância desta classe. Após o fechamento de uma instância desta classe todos os objetos persistidos por ela retornam ao estado oco, a não que esteja configurado para que sejam todos automaticamente destacados.

4.11.6.1 Armazenar objetos

Para armazenar um novo objeto no armazenamento de dados, deve-se usar o método *makePersistent()* do *PersistenceManager*, passando o objeto a ser persistido como parâmetro. Os objetos filhos armazenados nas suas propriedades são criados automaticamente nesta operação. Por fim este método retorna um objeto com seu estado marcado como persistente e sua chave preenchida.

O mesmo método responsável por criar novas entidades no armazenamento de dados é utilizado para atualizar objetos persistidos. Caso um objeto com seu estado marcado como persistido sujo, ou seja, um objeto persistido que possui propriedades alteradas, será atualizado no armazenamento de dados.

Caso a instância da classe *PersistenceManager* que manipulou o objeto esteja aberta após a alteração de dados das suas propriedades, não há a necessidade de usar o método *makePersistent()* para que as alterações sejam persistidas, pois isto é feito automaticamente no fechamento da instância.

4.11.6.2 Excluir objetos

A exclusão de objetos armazenados se dá de forma simples através da chamada ao método *deletePersistent()* do *PersistenceManager*, passando como parâmetro o objeto a ser excluído.

Caso o objeto excluído possua filhos de um relacionamento proprietário, estes também são excluídos em cascata.

4.11.6.3 Obter objetos

É possível obter um objeto persistido diretamente através da sua chave com o uso do método *getObjectById()* do *PersistenceManager*, passando como parâmetro o nome da classe do objeto em questão e a sua chave do tipo especificado no modelo do objeto.

É possível acessar todos os objetos de uma determinada classe com o uso do objeto *Extent* obtido pelo método *getExtent* do *PersistenceManager*, passando como parâmetro o nome da classe a ser recuperada. Isto cria um iterador capaz de acessar seqüencialmente todos os objetos da classe.

É possível o uso de *queries* para a obtenção de uma coleção de objetos. As consultas deste tipo devem ser em um tipo específico de entidade, podendo-se usar filtros e ordenação, através de uma consulta utilizando a linguagem de consulta *JDOQL* do JDO, semelhante à linguagem *SQL* utilizada em bancos de dados relacionais, porém muito mais restrita.

As consultas podem usar filtros contendo o nome do campo a ser filtrado, incluindo a chave, um operador e um valor. Vários filtros podem ser usados com operações de “e” lógico, porém operações de “ou” lógico não são suportadas. Ainda é possível especificar uma ordem de classificação dos resultados e um intervalo de consulta, começando do zero para o primeiro resultado.

```
Query query = pm.newQuery(VotingModel.class,"status == statusParam");
query.declareParameters("int statusParam");
result = (List<VotingModel>) query.execute(filter);
```

Figura 4.11: Exemplo de consulta utilizando JDO.

Há uma série de restrições às consultas *JDOQL* no App Engine. Uma delas especifica que um filtro sobre uma propriedade requer que esta propriedade exista no modelo da entidade. Outra restrição diz respeito ao uso de filtros de desigualdade (<, <=, >=, >), que não permite o uso de tais filtros em mais de uma propriedade, porém o permite com filtros de igualdade em outras propriedades, além das propriedades utilizadas com estes filtros requerem vir antes em ordens de classificação.

Para que uma propriedade possa ser utilizada em uma consulta ela deve possuir um índice definido.

4.11.7 Transações

Cada instância do *PersistenceManager* possui uma transação associada, obtida através do método *currentTransaction()*. O uso de transações permite a atomicidade das operações, ou seja, caso uma operação falhe, todas as outras dentro da transação não serão realizadas, além de garantir que entidades obtidas dentro da transação estarão atualizadas e consistentes.

Uma transação pode englobar operações sobre objetos de um mesmo grupo de entidades, ou seja, entidades que possuem uma entidade raiz em comum. Outra restrição diz que uma entidade não pode ser atualizada mais de uma vez numa mesma transação.

4.11.8 Objetos destacados

Objetos destacados são objetos persistentes desanexados da conexão com o armazenamento de dados, mas que mantêm todas as informações necessárias para uma posterior anexação, possibilitando assim, que sejam usados fora de uma transação ou depois que a instância da classe *PersistenceManager* que o recuperou seja fechada. Para que um objeto possa ser usado de forma destacada deve possuir isto explícito na sua modelagem.

Objetos destacados é uma ótima opção para serem usados como objetos de transferência de dados, já que podem ser destacados, transferidos, modificados e posteriormente re-anexados para sua atualização.

Por padrão o JDO implementa o padrão de projeto *Lazy Loading* (Carregamento Preguiçoso) para obter os valores das propriedades dos objetos. Isto significa que as propriedades são carregadas à medida que se requisita acesso a elas. Os objetos destacados, por não possuírem mais conexão com o armazenamento de dados não podem obter os valores das propriedades desta forma, fazendo então uso do recurso de grupo de busca, onde se especifica quais propriedades terão seu valor obtido ao se destacar o objeto. Por padrão este grupo se restringe às propriedades de primeiro nível, ou seja, propriedades das entidades filhas não são obtidas. O grupo de busca pode modificado através do uso do método *getFetchPlan()* do *PersistenceManager* para obter todas as propriedades existentes recursivamente.

4.12 Suporte a outras linguagens

O Google Application Engine para Java suporta que o código seja desenvolvido em outras linguagens que possuam um interpretador baseado na JVM, como é o caso do *Quercus*, que permite que código feito em *PHP* seja interpretado pela JVM, ou do *JRuby* que atua de forma semelhante para a linguagem *Ruby on Rails*.

Apesar destas possibilidades, a interação com o serviço de armazenamento de dados fica restrito à linguagem Java, devido à necessidade de uso de um dos seus padrões para persistência.

4.13 Conclusão

As facilidades de desenvolvimento em linguagem Java proporcionadas pelo Google Application Engine, conseguidas por meio da integração com a IDE de desenvolvimento Eclipse e kit de desenvolvimento local que simula de forma bastante fiel o ambiente real de execução do software, permitem que não haja mudanças drásticas nos processos de desenvolvimento de software em relação ao feito usualmente.

O maior esforço que se deve despender é em relação à adequação do projeto no que diz respeito a camada de persistência, onde deve ser projetado para atender ao mesmo aos padrões JDO ou JPA e as peculiares características do armazenamento de dados do Google Application Engine.

5. Google Web Toolkit

5.1 Introdução

O Google Web Toolkit, ou GWT, é uma poderosa ferramenta para auxiliar o desenvolvimento web. Com ela é possível a criação de aplicações para web com o código executado pelo cliente sendo feito em Java e trabalhar comunicações assíncronas com o servidor web, conhecidas como AJAX.

Para que isto possa acontecer, a ferramenta trabalha com o código dividido em duas partes distintas, uma com código a ser executado pelo cliente e outra com código a ser executado pelo servidor através de chamadas assíncronas feitas pelo código no cliente. O código Java destinado a ser executado pelo cliente é automaticamente compilado pelo Google Web Toolkit para código JavaScript capaz de ser executado por navegadores, deixando a cargo do compilador questões de compatibilidade entre navegadores e de otimização.

Aplicativos baseados neste conceito são muito mais agradáveis ao uso pelo cliente por evitar o uso excessivo de solicitações e se aproximar da interação proporcionada por aplicativos para desktop, evitando freqüentes carregamentos das páginas do aplicativo, melhorando a experiência de uso, algo que é um dos principais problemas de aplicativos web. O próprio Google utiliza esta ferramenta em vários de seus aplicativos, como no seu aplicativo de e-mail *Gmail*, a rede social *Orkut* e o conjunto de ferramentas como processadores de texto e planilhas do *Google Docs*.

Esta ferramenta pode ser utilizada por qualquer aplicativo Java para web e possui integração com o Google Application Engine sendo distribuída juntamente com seu plug-in para Eclipse.

5.2 Plug-in para Eclipse

O plug-in para Eclipse do Google Application Engine possui também integração para desenvolvimento de aplicações utilizando o GWT. O procedimento de instalação se dá do mesmo modo que a SDK do Google App Engine, devendo ser também selecionada para a instalação a SDK do Google Web Toolkit.

A criação de projetos que utilizam o GWT também decorre da mesma maneira que é feito para o App Engine. A estrutura do projeto se diferencia por estar divida em código do cliente e do servidor.

A execução de um projeto com GWT também é feita da mesma maneira descrita para o App Engine e todo seu código, tanto do cliente, como do servidor, é abarcado pelo depurador do Eclipse.

5.3 Chamadas RPC

É impossível de construir uma aplicação web robusta e que execute código apenas do lado do cliente. Por conta disso o GWT oferece um mecanismo capaz de executar código no lado do servidor, chamado de *RPC* (*Remote Procedure Calls*), ou chamadas de procedimento remoto, que são invocadas pelos mecanismos GWT através de solicitações assíncronas ao servidor.

O código presente em classes que são executadas em solicitações *RPC* não são compilados para a linguagem *JavaScript*, já que são executados pelo servidor web, podendo portanto fazer uso de qualquer biblioteca Java.

5.3.1 Criando serviços RPC

Para se criar um serviço *RPC* é necessário definir uma interface para o serviço que estenda a interface *RemoteService* do GWT , definir uma interface

assíncrona e implementar uma classe que estenda *RemoteServiceServlet* com o código efetivamente.

Devido às características das chamadas assíncronas, as quais não podem ser bloqueadas durante sua execução, seus da interface assíncrona não podem retornar um resultado, sendo esta peculiaridade contornada com o uso de funções *callback*, que são chamadas ao final da execução do código com os resultados.

```
@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {
  String greetServer(String name);
}
```

Figura 5.1: Definição da interface do serviço.

```
public interface GreetingServiceAsync {
  void greetServer(String input, AsyncCallback<String> callback);
}
```

Figura 5.2: Definição da interface assíncrona do serviço.

```
public class GreetingServiceImpl extends RemoteServiceServlet implements
    GreetingService {

  public String greetServer(String input) {
    return "";
  }
}
```

Figura 5.3: Implementação do serviço RPC.

Para que o compilador GWT interprete corretamente o código, é necessário que o nome da interface assíncrona seja o nome utilizado na interface do serviço adicionado do sufixo *Async* além de estarem no mesmo pacote. Além disso, vale ressaltar que nos métodos definidos na interface assíncrona há o parâmetro com a função de retorno, que deve ser do tipo *AsyncCallback<tipo>*, onde *tipo* é o tipo de dados retornado.

Vale ressaltar a existência da anotação *RemoteServiceRelativePath("greet")* na figura Figura 5.1: Definição da interface do serviço.. Esta anotação serve para determinar a URL que deve ser definida para o servlet da implementação do serviço RPC definida no descritor de implementação *web.xml*. A URL será definida pelo nome do módulo (o qual é detalhado adiante) e o nome passado como parâmetro para esta anotação.

```
<servlet>
  <servlet-name>greetServlet</servlet-name>
  <servlet-class>sample.server.GreetingServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>greetServlet</servlet-name>
  <url-pattern>/stockwatcher/greet</url-pattern>
</servlet-mapping>
```

Figura 5.4: Definição do servlet da implementação RPC no descritor de implementação do aplicativo.

5.3.2 Realizando chamadas RPC

Para realizar uma chamada RPC no código do cliente, é necessário instanciar a interface do serviço através do método *GWT.create()* para posteriormente chamar seus métodos de acordo com a assinatura disponibilizada pela interface assíncrona.

```
GreetingServiceAsync greetingService = GWT
.create(GreetingService.class);

greetingService.greetServer(textToServer,
        new AsyncCallback<String>() {
            public void onFailure(Throwable caught) {
                // Falha
            }

            public void onSuccess(String result) {
                // Sucesso
            }
        });
```

Figura 5.5: Realização de uma chamada RPC.

Nota-se através da figura Figura 5.5 que a função de retorno deve implementar um método para o caso de falha na chamada e outro para o caso de sucesso.

5.3.3 Serialização

Serialização é importante para realizar a transferência de dados entre o servidor e o cliente nas chamadas RPC. Um tipo é serializável se uma ou mais das seguintes condições forem verdadeiras:

a) O tipo é primitivo, como por exemplo, *byte* ou *int*.
b) O tipo é uma instância de *String* ou *Date*.
c) O tipo é uma enumeração.
d) O tipo é um *array* ou coleção de tipos serializáveis.
e) O tipo é uma classe definida pelo usuário que implementa a interface *java.io.Serializable* ou *IsSerializable* do GWT sem construtores definidos, ou com pelo menos um construtor sem argumentos.
f) O tipo tem ao menos uma subclasse serializável.

Nas versões anteriores a 2.0, o GWT não permitia a transmissão de objetos de classes aprimoradas pelo JDO com adição de informações extras, necessitando de técnicas, como o uso de objetos de transferência de dados ou frameworks capazes de realizar esta tarefa, para contornar esta situação. Entretanto a partir da versão 2.0 o GWT é capaz de realizar a serialização completa destes objetos desde que estejam destacados do sistema de armazenamento de dados.

5.4 Construindo aplicações

Uma aplicação que utiliza o GWT basicamente possui código a ser executado pelo cliente, código a ser executado pelo servidor, além de definições de módulos e arquivos estáticos.

A aplicação deve possuir um arquivo HTML contendo a seguinte etiqueta HTML:

- *<script language="javascript" src="modulo/modulo.nocache.js">*
  </script>

A etiqueta acima significa que um módulo GWT compilado para JavaScript deve ser carregado na página, onde *modulo* é o nome do módulo GWT. Além da etiqueta anterior, este arquivo HTML, chamado de *host page*, deve possuir um container com um identificador atribuído para ser capaz de abrigar elementos visuais do GWT.

5.4.1 Módulos

As aplicações GWT são divididas em uma ou mais partes chamadas de módulos. Cada módulo deve ser definido na raiz do seu pacote através de um arquivo de configuração XML com o nome do módulo acrescido do sufixo ”.*gwt.xml*”.

O arquivo de configuração de um módulo deve definir seu nome, uma classe entrada para ser instanciada quando o módulo é carregado, sub-pacotes que contenham código que deva ser compilado para *JavaScript* e sub-pacotes públicos que contenham arquivos estáticos.

Caso as classes de um módulo utilizem código presente em outro módulo, este deve ser especificado no arquivo de configuração através da etiqueta <*inherits*>.

5.4.2 Classe de entrada

Uma classe de entrada deve ser definida como tal no arquivo de configuração do módulo e implementar a interface *EntryPoint*. Quando uma classe de entrada é instanciada o seu método *onModuleLoad()* é chamado, dando ínicio à execução do código criado pelo GWT.

5.4.3 Interface gráfica

O GWT disponibiliza uma série de componentes para criar a interface gráfica da aplicação GWT chamados de *widgets*. A interface gráfica é construída adicionando-se *widgets* aos containers criados no arquivo HTML que contém o módulo.

5.4.4 Executando uma aplicação

Uma aplicação construída com o uso do GWT é executada quando uma página HTML que hospeda um de seus módulos que tenham uma classe de entrada definida é acessada fazendo seu código *JavaScript* ser executado.

5.4.5 Conclusão

O uso de ferramentas como o Google Web Toolkit pode facilitar enormemente a criação de aplicativos mais dinâmicos, intuitivos e agradáveis ao uso, utilizando *JavaScript*, *AJAX* e Java, sem que o desenvolvedor se preocupe com aspectos desagradáveis, como integração ou compatibilidade entre navegadores. Além disso, pode ser uma forma interessante de diminuir a carga de código a ser processada pelo servidor web, distribuindo-a com o cliente. Por sua capacidade de integração com o Google App Engine, pode ser uma forma poderosa para o desenvolvimento de aplicações voltadas a este serviço.

6. Exemplo de aplicação

6.1 Introdução

Com o objetivo de completar os estudos acerca das tecnologias apresentadas, foi desenvolvida uma aplicação capaz de trabalhar com alguns dos aspectos apresentados.

Trata-se de um sistema de votação desenvolvido para web com o auxílio da ferramenta Google web Toolkit, através do qual seja possível criar votações, votar em uma votação e retirar relatórios das votações. Tudo isto deve funcionar hospedado e executado sob a estrutura do Google Application Engine.

6.2 Ferramentas de suporte

Para a realização do projeto foi utilizada a tecnologia Java para desenvolvimento de aplicações web através da versão 6 do seu kit de desenvolvimento *Java2EE* e versão 1.3.3 do kit de desenvolvimento do Google Web Toolkit. Como suporte foi utilizado a ferramenta Google Web Toolkit versão 2.03, o ambiente de desenvolvimento Eclipse e ferramentas de controle de versão.

6.2.1 Eclipse

A aplicação foi desenvolvida fazendo-se uso do ambiente de desenvolvimento integrado Eclipse para Java *Enterprise Edition*, na sua versão *Galileo 3.5*. Para completar o desenvolvimento foi utilizado o plug-in para Eclipse contendo o kit de desenvolvimento do Google Application Engine na versão 1.3.3 e Google Web Toolkit na versão 2.03, obtido como descrito no capítulo 4, onde este assunto é abordado.

6.2.2 SVN

Para o desenvolvimento do projeto se recorreu ao auxílio do sistema de controle de versão *Subversion* ou *SVN*. O controlador escolhido foi o plug-in para eclipse conhecido como *Subversive*. O repositório utlizado foi o disponibilizado pela rede *inf* da UFSC localizado no seguinte endereço:

- HTTPS://svn.inf.ufsc.br/vdmuller

6.3 Especificações

Este item aborda as especificações necessárias para o projeto da aplicação proposto, como seus requisitos.

6.3.1 Requisitos funcionais

A aplicação tem definidos os seguintes requisitos funcionais:

a) Deve ser possível a um administrador da aplicação, criar, excluir, editar ou obter relatórios de votações.
b) Deve ser possível iniciar e finalizar votações.
c) O sistema deve notificar os votantes do inicio da votação através de uma mensagem de e-mail.
d) Deve ser possível votar em uma votação.
e) Deve ser possível cadastrar auditores para uma votação.
f) Os auditores podem obter relatórios de uma votação.

6.3.2 Requisitos não funcionais

A aplicação possui ainda os alguns dos seus requisitos não funcionais listados a seguir:

a) O administrador da aplicação deve se autenticar através da sua conta do Google para administrar as votações.
b) Os auditores são convidados por meio do e-mail da sua conta do Google.
c) Os auditores devem se autenticar com sua conta do Google para obter relatórios da votação.
d) Ao se criar ou editar uma votação, deve realizar a validação dos campos.
e) Não é possível excluir uma votação em andamento.
f) A votação se dá por um link recebido através de mensagem de e-mail pelo votante.
g) O sistema deve informar o resultado da votação aos votantes através de uma mensagem de e-mail quando a votação for finalizada.
h) A interface gráfica deve ser feita utilizando a ferramenta Google Web Toolkit.

6.4 A aplicação

Para atender às especificações propostas a aplicação foi divida em três módulos distintos, um para cada papel nelas descritos, responsáveis por todos os processos necessários para uma votação. Cada módulo é implementado com o uso das técnicas fornecidas pelo Google Web Toolkit, possuindo sua interface gráfica totalmente feita com o uso de *widgets* do GWT a fim de possibilitar uma experiência de uso agradável ao usuário, com respostas rápidas e acesso assíncrono ao servidor da aplicação, contornando a sensação incômoda gerada pelo uso de aplicativos que não fazem uso de tecnologia semelhante. Cada módulo compartilha com os demais o mesmo modelo de dados definido pela aplicação e é acessado por meio da página HTML que o hospeda.

6.4.1 Arquitetura

A arquitetura da aplicação se baseia na mescla dos padrões de projeto MVC (*Model View Controller)* e *Mediator* adaptados às necessidades causadas pelas características de divisão de código entre cliente e servidor do Google Web Toolkit.

O MVC é o padrão de projetos mais utilizado para o desenvolvimento de aplicação web. Este padrão determina que a aplicação esteja divida em três partes ou camadas com especialidades distintas. O modelo é responsável por representar os dados da aplicação, realizando o acesso ao sistema de armazenamento e lógica sobre eles. A visão trata de como o modelo é apresentado ao usuário. Por fim, a camada de controle recebe todas as solicitações da aplicação e é responsável por executar operações sobre o modelo e a visão.

No padrão de projetos *Mediator*, ou mediador, uma classe torna-se responsável por fazer a ligação com todas as demais, com estas fazendo suas solicitações à classe mediadora, resultando num menor acoplamento entre classes.

A aplicação desenvolvida possui uma camada responsável pelo modelo de dados. Esta camada é constituída pelos modelos de objetos JDO e uma classe de acesso aos dados.

Cada módulo da aplicação possui uma classe responsável pelo controle, caracterizada pela classe que atende às solicitações RPC do GWT encarregada de operar o modelo.

A visão de cada módulo é feita toda com o uso do GWT e implementa o padrão mediador, onde a classe de entrada se torna o mediador de todos os outros componentes da visão.

6.4.2 Modelo de dados

O modelo de dados da aplicação se baseia em quatro entidades modeladas de acordo como descrito pelo JDO, além de uma classe responsável pelo acesso aos dados.

A entidade *VotingModel* representa uma votação e é responsável por armazenar todos os dados referentes a ela. Esta entidade possui as seguintes porpriedades:

a) *id* – Chave na forma de *String* codificada.
b) *title* – *String* que armazena o título da votação.
c) *description* – *String* que armazena a descrição da votação.
d) *type* – *Enum* que indica se uma votação é uma eleição ou uma enquete.
e) *status* – *Enum* que indica o estado da votação, o qual pode ser de esperando iniciar, ocorrendo ou finalizada.
f) *date* – *Date* que contém a data de criação da votação pelo administrador.
g) *start* – *Date* que indica quando a votação deve ser iniciada.
h) *end* – *Date* que indica quando a votação deve ser encerrada.
i) *voters* – *List* que define uma relação proprietária bilateral de um-para-vários com a entidade *Voter*.
j) *options* – *List* que define uma relação proprietária bilateral de um-para-vários com a entidade *Options*.
k) *auditors* – *List* que contém o e-mail dos auditores da votação.

A entidade *Voter* representa um votante capaz de votar em alguma votação. Suas propriedades são as seguintes:

a) *id* – Chave na forma de *String* codificada.
b) *name* – String que contém o nome do votante.
c) *email* – String que contém o e-mail do votante
d) *votingModel* – Objeto *VotingModel* que referencia a votação de qual o votante faz parte.

e) *vote* – Objeto *Vote* que define uma relação proprietária bilateral de um-para-um com a entidade que armazena o voto do votante.

Quando um votante vota em uma votação, seu voto é armazenado na forma de um objeto da entidade *Voter* através das seguintes propriedades:

a) *id* – Chave na forma de *String* codificada.
b) *voter* – Objeto *Voter* que referencia o votante dono do voto.
c) *option* – *String* que armazena a chave da opção votada no caso da votação ser uma enquete.
d) *nominee* – *String* que armazena a chave de um *Voter* canditato em uma votação do tipo eleição.

No caso de uma votação ser do tipo enquete, ela deve possuir as opções de voto na forma de uma entidade do tipo *Option*, que armazena os dados da opção com as seguintes propriedades:

a) *id* – Chave na forma de *String* codificada.
b) *title* – *String* que armazena o título da opção.
c) *description* – *String* que armazena a descrição da opção.
d) *votingModel* – Objeto *VotingModel* que referencia a votação de qual a opção faz parte.

Para o acesso da aplicação ao serviço de armazenamento de dados, existe uma classe que implementa o padrão de acesso à camada de persistência *DAO (Data Access Object),* onde há uma classe com funções que manipulam as entidades através de objetos de acesso ao armazenamento de dados do JDO. Esta classe apresenta funcionalidades como a de persistir objetos, recuperar objetos ou retornar uma coleção de objetos através de uma consulta. Os controladores de cada módulo devem utilizar um objeto desta classe para realizarem operações com o modelo de dados.

6.4.3 Módulo de administração

O módulo administrativo reúne as funções destinadas aos administradores do sistema. Para acessá-lo é necessário que um administrador da aplicação se autentique com sua conta do Google, podendo então ver a lista de votações criadas e criar, excluir, editar ou ver relatórios de votações.

A implementação deste módulo é feita com o uso do GWT e foi realizada de modo que a classe de entrada do módulo sirva de mediadora para as demais, ou seja, ela é a responsável por se comunicar com todos os demais componentes da interface gráfica e realizar chamadas RPC.

Sua implementação do serviço RPC trata de realizar as operações necessárias sobre as entidades para que os requisitos sejam cumpridos, além de permitir a sua transferência para a camada do cliente.

6.4.4 Módulo de auditoria

O módulo de auditoria reúne as funções destinadas aos auditores de uma votação. Para acessá-lo é necessário que o auditor de uma votação se autentique com sua conta do Google, podendo então ver a lista de votações que audita, podendo ver seus relatórios.

A implementação deste módulo é feita com o uso do GWT e foi realizada de modo que a classe de entrada do módulo sirva de mediadora para as demais, ou seja, ela é a responsável por se comunicar com todos os demais componentes da interface gráfica e realizar chamadas RPC.

Sua implementação do serviço RPC trata de realizar as operações necessárias sobre as entidades para que os requisitos sejam cumpridos, além de permitir a sua transferência para a camada do cliente.

6.4.5 Módulo de votação

O módulo de votação permite que o votante de uma votação vote através de uma interface gráfica simples que mostra as informações necessárias para o voto. Este módulo é implementado como os módulos anteriores, comunicando-se com o seu serviço RPC para obter os dados da votação referente ao votante ou enviar as informações de voto, além de verificar se estas informações são válidas.

6.4.6 Tarefas Agendadas

A aplicação utiliza o recurso de tarefas agendas para iniciar ou finalizar votações. É definida uma tarefa a ser executada a cada minuto com a função de verificar quais votações devem ser inicializadas e quais devem ser finalizadas naquele momento.

Para que isto ocorra é implementado um servlet de nome *VotingCron* configurado no descritor de implementação com acesso restrito à administradores, capaz de iniciar ou finalizar votações.

Assim que uma votação é iniciada, o servlet manda um e-mail para todos os votantes informando o inicio da votação e um endereço para votar. Do mesmo modo, quando uma votação é finalizada, o votantes tornam a receber um e-mail informando o seu fim.

6.4.7 Desafios

Provavelmente o maior desafio ao se implementar aplicação se deu ao modelar de forma eficiente as entidades persistentes. Lidar com questões de chaves, grupos de entidades, relacionamentos e, sobretudo o desafio de transferir dados de forma eficiente entre as camadas do servidor e do cliente.

Com o uso de versões do GWT anteriores a 2.0, era impossível a transferência direta de objetos aprimorados do JDO, necessitando uma solução para contornar a questão.

Inicialmente foi levantado o uso do padrão para transferência de dados entre camadas conhecido como DTO (*Data Transfer Objects*), porém esta solução a que melhor se adaptava ao problema, pois necessitava que para cada entidade se criasse um objeto de transferência de dados com suas propriedades para transferência, além de funções que também transferissem os dados da entidade para o seu objeto de transferência de dados e vice-versa.

Depois de muito procurar uma solução, foi feito uso do framework *Gilead*, comumente utilizado para realizar estas transferências de dados entre o GWT e objetos de persistência do framework de persistência Java *Hibernate*, que possui uma versão experimental para suporta a persistência do Google Application Engine, conhecida como *Adapter4AppEngine*. Este framework mostrou-se bastante satisfatório atendendo ‘as questões da aplicação proposta.

Porém com a versão 2.0 do GWT vem também o suporte à transferência de uma forma natural através da completa serialização de objetos melhorados pelo JDO do App Engine destacados, possibilitando sua transferência, modificação pro parte do código do cliente e atualização sem maiores esforços, dispensando qualquer framework relativo.

6.5 Processo de votação

O processo de votação inicia com um administrador que ao se autenticar no sistema tem a capacidade de criar uma votação. Para criar uma votação é necessário acessar a opção referente, preencher os dados da votação na tela que surgirá. Após a correta validação dos dados um e-mail é enviado para cada votante informando-o da sua inclusão.

No horário programado a votação é iniciada e um e-mail contendo um endereço individual para votar é enviado para cada votante. Ao acessar o endereço o votante escolhe sua opção preferida e realiza a votação através de um clique em um botão.

Ao ser finalizada a votação, a aplicação envia um e-mail contendo o resultao final para os votantes.

6.6 Conclusão

O desenvolvimento da aplicação teve enorme valor para consolidar os ensinamentos adquiridos com o decorrer do trabalho, sendo uma grande oportunidade de avaliar o ambiente de desenvolvimento local e a integração com as ferramentas que complementam o App Engine, como o GWT.

Esta aplicação serviu para constatar que o ponto forte do desenvolvimento para o Google Application Engine está presente no seu completo e robusto ambiente de desenvolvimento local aliado ao plug-in para Eclipse e interação com ferramentas de suporte como o GWT.

7. Discussão

7.1 Introdução

Uma das áreas da computação em que mais se tem avançado nos últimos anos diz respeito à web. O crescente desenvolvimento de tecnologias nesta área permitiu que vários aspectos relacionados ao desenvolvimento de aplicações e infraestrutura de hardware fossem repensados.

Com o avanço obtido com as tecnologias relacionadas à internet nos últimos anos, o desenvolvimento de aplicações web tem ganhado cada vez mais força, caracterizando-se como uma tendência para o desenvolvimento de aplicações. O suporte de novas tecnologias relacionadas a hardware, sistemas operacionais e virtualização, propiciou que o modo de disponibilizar recursos de hardware através da internet fosse repensado, e conseqüentemente o modo de disponibilizar e desenvolver aplicações.

O agrupamento destes ramos em comum, acarretando no surgimento do paradigma da computação em nuvem, gerou um novo e fértil campo de pesquisa, possibilitando que essas áreas tão próximas se desenvolvam de maneira conjunta, potencializando seu desenvolvimento. A partir daí surge a tendência de se criar aplicações que façam uso de escalabilidade, tornando mais eficiente a alocação de recursos e eliminando barreiras para o seu contínuo crescimento.

Várias grandes empresas de tecnologia vêm desenvolvendo soluções nesta área. O Google saiu na frente ao desenvolver um ambiente completo de desenvolvimento com uma grande abrangência das áreas da computação em nuvem. A empresa de vendas pela internet *Amazon.com* também foi uma das pioneiras na disponibilização destes serviços, desenvolvendo o *Amazon Elastic Compute Cloud* ou *EC2*, ao procurar uma forma de utilizar todo seu poder computacional fora dos momentos de pico de uso do seu sistema. A *Microsoft*

também aderiu a este paradigma, primeiramente disponibilizando alguns de seus softwares como serviço e posteriormente, serviços de plataforma e infraestrutura.

Através da ferramenta *Google Insights*, que retorna um gráfico com a proporção de pesquisas de um termo em relação aos anos, mostra o interesse relativamente novo nos termos "*Cloud Computing*" e "*App Engine*".

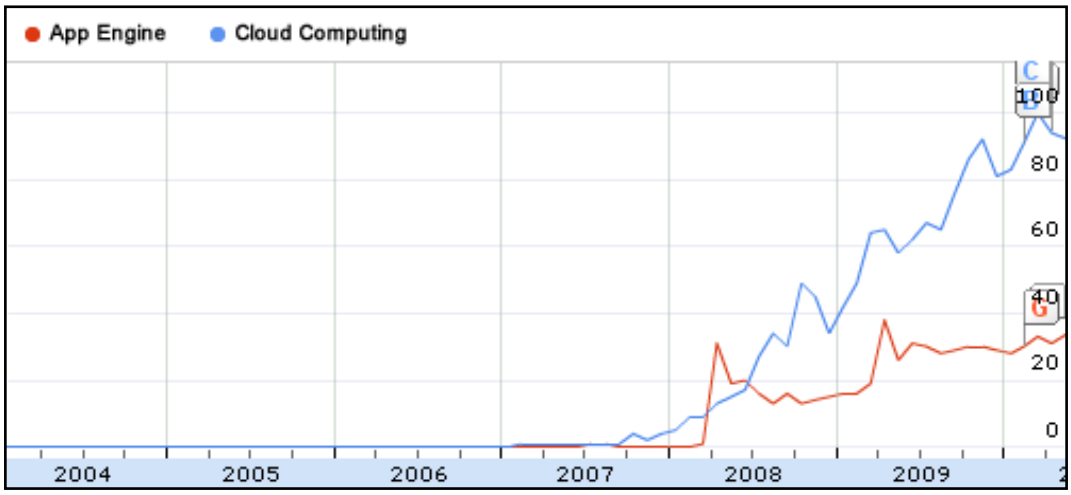

Figura 7.1: Pesquisas dos termo Cloud Computing e App Engine no Google Insights.

Por conta disso se verifica o pouco material disponível sobre o assunto, o que por outro lado abre oportunidades para o desenvolvimento de pesquisas na área.

7.2 Google Application Engine

O Google se destacou ao desenvolver o Google Application Engine como serviço pioneiro, que dispõe ao mesmo tempo de plataforma e de infraestrutura como serviços para a criação aplicações web.

A grande inovação, e ponto forte deste serviço está em por em prática um conjunto dos conceitos da computação em nuvem ao providenciar uma completa plataforma para desenvolvimento de aplicações, contando com kit de desenvolvimento e ambiente de execução libertando o desenvolvedor de maiores preocupações técnicas, ambos automaticamente escaláveis. Seu ponto chave está

em a aplicação poder atender uma crescente demanda de solicitações e armazenamento.

Entretanto, a série de limites que seu ambiente de execução impõe impede que se faça uso de todo o potencial de escalabilidade que o App Engine é capaz de alcançar. A principal desvantagem está relacionada ao seu sistema de armazenamento de dados, que apesar de todo seu potencial possui uma série de restrições, como por exemplo, o tamanho máximo que uma entidade pode possuir. Para tentar contornar os inconvenientes destes limites o Google providencia alguns recursos na forma de serviços.

Há também a necessidade de disponibilizar uma maior gama de recursos faturáveis, além dos essenciais ao atendimento de qualquer solicitação. A maioria dos recursos está sujeita a cotas que não são passíveis de serem controladas pelo desenvolvedor. Outro ponto a considerar está na incapacidade em poder adquirir maior poder de processamento ou de recursos de paralelismo a fim de tornar mais rápida a execução de uma solicitação, pois esta classe de recursos só está destinada ao atendimento de um maior número de solicitações.

Por ambiente de execução impor uma série de limitações, dificuldade de portabilidade, por se tratar de uma arquitetura específica, e a incapacidade de migração dos dados armazenados pela aplicação não é encorajador o projeto de grandes aplicações baseadas neste sistema, pois implicaria na sua dependência ao App Engine. O que parece mais indicado é o desenvolvimento de pequenas aplicações que podem ir crescendo gradualmente, inicialmente fazendo uso dos recursos disponibilizados gratuitamente.

É importante ressaltar que por se tratar de uma ferramenta nova e em constante desenvolvimento, seu amadurecimento pode sanar as dificuldades apresentadas. Um exemplo disso é o desenvolvimento do App Engine para negócios, destinado a empresas, com previsão de implementação de serviços de banco de dados em SQL e conexões SSL em todos os domínios da aplicação.

7.3 Google Web Toolkit

O Google Web Toolkit mostrou-se uma ferramenta com enorme potencial para facilitar a criação de aplicações com interfaces ricas integradas ao Java. Esta ferramenta possibilita através da sua integração com a linguagem Java, que se desenvolvam aplicações que utilizam ao máximo os recursos que os navegadores disponibilizam para a criação de aplicações web modernas.

7.4 Conclusão

Apesar das limitações que o Application Engine possui, esta é uma ferramenta com enorme potencial de desenvolvimento e que molda a tendência do futuro para o desenvolvimento web.

8. Considerações finais

Este estudo cumpriu os objetivos gerais ao ser feita uma análise minuciosa da documentação da documentação da ferramenta App Engine do Google, apontando suas principais características, e pondo em prática os estudos através da elaboração de uma aplicação de exemplo.

Com o cumprimento do objetivo geral naturalmente os objetivos específicos se concretizaram através da análise do Google Application Engine, seu caso específico com a linguagem Java, análise do Google Web Toolkit, o qual pode servir de suporte para o anterior e, por fim, o que se pôde experimentar com o desenvolvimento da aplicação de suporte.

Como sugestões para trabalhos futuros, ficam os seguintes temas:

a) Elaborar uma análise de desempenho das ferramentas.
b) Elaborar comparativos com ferramentas afins.
c) Realizar estudos relacionados aos aspectos de segurança da ferramenta.
d) Projetar e implementar ambientes baseados nos paradigmas da computação em nuvem.

9. Referências Bibliográficas

AKRAM, Tahir. *Developing Java Based Web Applications in Google App Engine.* 19 de Dezembro de 2009. http://www.uet.edu.pk/Conferences/icosst2009/presentations_2009/OSSW_Presentations/2_Developing_Java_Based_Web_Applications_in_Google_App_OSSW_ICOSST_2009.pdf (acesso em Fevereiro de 2010).

ALECRIM, Emerson. "O que é Cloud Computing (Computação nas Nuvens)?" *InfoWester.* 23 de Dezembro de 2008. http://www.infowester.com/cloudcomputing.php (acesso em Março de 2010).

ANDERSON, Chris. *A cauda longa: do mercado de massa para o mercado de nicho.* Rio de Janeiro: Elsevier, 2006.

APACHE. *Java Data Objects (JDO).* http://db.apache.org/jdo/ (acesso em Abril de 2010).

BARBOSA, Fernando P., e Andrea S. CHARAO. "Grid Computing e Cloud Computing – Uma Relação de Diferenças, Semelhanças, Aplicabilidade e Possibilidades de Cooperação entre os dois Paradigmas." *Programa de Pós-Graduação em Informática – Universidade Federal de Santa Maria.*

BARROS, Fabio. "Cloud Computing: Prepare-se para a nova onda em tecnologia." *COMPUTERWORLD.* 17 de Abril de 2008. http://computerworld.uol.com.br/gestao/2008/04/17/cloud-computing-prepare-se-para-a-nova-onda-em-tecnologia/ (acesso em Fevereiro de 2010).

BOBZIN, Heiko et al. "Getting Started with JDO." *InformIT.* 27 de Fevereiro de 2004. http://www.informit.com/articles/article.aspx?p=169515 (acesso em Maio de 2010).

CARR, David F. "How Google Works." *Baselinemag.* 06 de 07 de 2006. http://www.baselinemag.com/c/a/Infrastructure/How-Google-Works-1/ (acesso em Março de 2010).

CHANG, Fay et al. "Bigtable: A Distributed Storage System for Structured Data." *OSDI*, 2006: 14.

CHONG, Frederick, e Gianpaolo CARRARO. "Architecture Strategies for Catching the Long Tail." *Microsoft Corporation.* Abril de 2006. http://msdn.microsoft.com/en-us/library/aa479069.aspx (acesso em Abril de 2010).

“Cloud computing, a nuvem que se aproxima.” 9 de Dezembro de 2008. http://camorim.wordpress.com/2008/12/09/cloud-computing-a-nuvem-que-se-aproxima/ (acesso em Fevereiro de 2010).

COWARD, Danny, e Yutaka YOSHIDA. *Java™ Servlet Specification Version 2.4.* 24 de Novembro de 2003.

CROMWELL, Ray. “Google AppEngine and GWT now a marriage made in heaven.” 2009 de Abril de 2009. http://timepedia.blogspot.com/2009/04/google-appengine-and-gwt-now-marriage.html (acesso em Março de 2010).

FOSTER, Ian, Yong ZHAO, Ioan RAICU, e Shiyong LU. “Cloud Computing and Grid Computing 360-Degree Compared.” *Department of Computer Science, University of Chicago.*

GHEMAWAT, Sanjay, Howard GOBIOFF, e Shun-Tak LEUNG. “The Google File System.” *SOSP*, Outubro de 2003: 15.

“GOOGLE ARCHITECTURE.” *High Scalability.* 22 de Novembro de 2008. http://highscalability.com/google-architecture (acesso em Março de 2010).

GOOGLE. “Guia do desenvolvedor.” *Google App Engine.* 2010. http://code.google.com/intl/pt-BR/appengine/docs/ (acesso em Maio de 2010).

—. “Product Overview.” *Google Web Toolkit.* 2010. http://code.google.com/intl/pt-BR/webtoolkit/overview.html (acesso em Maio de 2010).

GREENE, Kate. “Google's Cloud Looms Large.” *Technology Review.* 3 de Dezembro de 2007. http://www.technologyreview.com/Biztech/19785/?nlid=701&a=f (acesso em Fevereiro de 2010).

HAUSMAN, Marc. “The Great SaaS Debate.” 19 de Janeiro de 2008. http://strategicguy.blogspot.com/2008/01/great-saas-debate.html (acesso em Fevereiro de 2010).

“How to Understand AppEngine DataStore Under the Hood.” 11 de Agosto de 2008. http://techblog.ironfroggy.com/2008/08/how-to-understand-appengine-datastore.html (acesso em Fevereiro de 2010).

KEENE, Christopher. “What Is Platform as a Service (PaaS)?” 18 de Março de 2009. http://www.keeneview.com/2009/03/what-is-platform-as-service-paas.html (acesso em Fevereiro de 2010).

KNORR, Eric, e Galen GRUMAN. “What cloud computing really means.” *InfoWorld.* http://www.infoworld.com/d/cloud-computing/first-look-googles-high-flying-cloud-python-code-190 (acesso em Fevereiro de 2010).

MAHMOUD, Qusay H. “Getting Started With Java Data Objects (JDO): A Standard Mechanism for Persisting Plain Java Technology Objects.” *Sun Developer Network.* 9 de Agosto de 2005. http://java.sun.com/developer/technicalArticles/J2SE/jdo/ (acesso em Abril de 2010).

MILLER, Robert J. “Google App Engine for Java - 3 Tips for Getting Started.” http://www.pardontheinformation.com/2009/06/google-app-engine-for-java-3-tips-for.html (acesso em Fevereiro de 2010).

MOREIRA, Daniela. “Cloud computing: entenda este novo modelo de computação.” *IDGNow.* 12 de Agosto de 2008. http://idgnow.uol.com.br/computacao_corporativa/2008/08/13/cloud-computing-entenda-este-novo-modelo-de-computacao/paginador/pagina_4 (acesso em Fevereiro de 2010).

“O que é cloud computing?” *CIO.* 24 de Julho de 2008. http://cio.uol.com.br/tecnologia/2008/07/24/o-que-e-cloud-computing/ (acesso em Fevereiro de 2010).

PACHECO, Diego. “PaaS, Cloud Computing, Virtualização e o Futuro.” *iMasters.* 4 de Setembro de 2009. http://imasters.uol.com.br/artigo/14165/gerenciadeprojetos/paas_cloud_computing_virtualizacao_e_o_futuro_parte_01/ (acesso em Fevereiro de 2010).

“Reviewing Google AppEngine for Java (Part 2).” 6 de Junho de 2009. http://blog.newsplore.com/2009/06/06/reviewing-google-appengine-for-java-part-2/comment-page-1 (acesso em Dezembro de 2009).

ROOS, Robin M. “JDO Architecture.” *InformIT.* 16 de Janeiro de 2004. http://www.informit.com/articles/article.aspx?p=102306&seqNum=4 (acesso em Maio de 2010).

SUN MICROSYSTEMS. “Designing Enterprise Applications with the J2EETM Platform, Second Edition.” http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html#1080752 (acesso em Janeiro de 2010).

—. “Front Controller.” 2002. http://java.sun.com/blueprints/patterns/FrontController.html (acesso em Janeiro de 2010).

—. “INTRODUCTION TO THE JAVA™ EE 6 PLATFORM.” Dezembro de 2009: 18.

WAYNER, Peter. “Cloud versus cloud: A guided tour of Amazon, Google, AppNexus, and GoGrid.” *InfoWorld.* 21 de Julho de 2008. http://www.infoworld.com/d/cloud-computing/cloud-versus-cloud-guided-tour-amazon-google-appnexus-and-gogrid-122 (acesso em Fevereiro de 2010).

—. “First look: Google's high-flying cloud for Python code.” *InfoWorld.* 12 de Maio de 2008. http://www.infoworld.com/d/cloud-computing/first-look-googles-high-flying-cloud-python-code-190 (acesso em Fevereiro de 2010).

WIKIPEDIA. *Cloud computing.* http://en.wikipedia.org/wiki/Cloud_computing (acesso em Fevereiro de 2010).

—. “Google App Engine.” http://en.wikipedia.org/wiki/Google_App_Engine (acesso em Março de 2010).

—. “Google File System.” http://en.wikipedia.org/wiki/Google_File_System (acesso em Abril de 2010).

Wikipedia. “MapReduce.” http://en.wikipedia.org/wiki/MapReduce (acesso em Abril de 2010).

10. Anexos

10.1 Anexo A – Artigo

Desenvolvimento de aplicações sob o paradigma da computação em nuvem com ferramentas Google

Victor Daniel Müller

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

Caixa Postal 475 – 88040-900 – Florianópolis – SC – Brasil

vdmuller@inf.ufsc.br

Abstract. *The increasing progress in the development of web technology has enabled a new approach to web development, named cloud computing, awakening to a new possibility of development and distribution of applications. Based on this approach, an analysis is made of the tools provided by Google through the Google App Engine and Java for development of scalable applications, and tools that serve as aid for web applications like these.*

Resumo. *O crescente avanço no desenvolvimento das tecnologias web possibilitou uma nova abordagem sobre o desenvolvimento web, nomeada de computação em nuvem, despertando para uma nova possibilidade de desenvolvimento e distribuição de aplicações. Baseado nesta abordagem é feita uma análise das ferramentas disponibilizadas pelo Google através do Google App Engine e Java para desenvolvimento de aplicações escaláveis, além de ferramentas que sirvam de auxílio para aplicações web como estas.*

1. Introdução

Com a evolução de diversas tecnologias computacionais correlatas, como a computação distribuída, internet e linguagens de programação, tornou-se possível o surgimento e exploração de novas áreas da computação.

A idéia de vender recursos computacionais sob demanda, de acordo com a necessidade do cliente é uma idéia antiga, porém se tornava limitada às grandes instituições capazes de adquirir recursos computacionais das grandes detentoras de poder computacional.

Com o passar do tempo, e conseqüente desenvolvimento de novas tecnologias nos mais abrangentes campos da computação, principalmente as relacionadas à massificação do uso da internet a idéia voltou à tona. Primeiramente procurou-se disponibilizar aplicativos web que poderiam ser acessados de qualquer lugar através da internet. Mais adiante, o desenvolvimento de tecnologias de virtualização viabilizou a venda sob demanda e de forma escalável de infraestrutura e recursos computacionais capazes de sustentar estes aplicativos web.

O agrupamento destas áreas sob uma nova ótica fez surgir o paradigma da computação em nuvem, gerando a tendência cada vez maior de aplicativos que possam ser acessados de forma eficiente de qualquer lugar. Este paradigma criou a necessidade de repensar o modo como as aplicações são desenvolvidas e disponibilizadas, ao mesmo tempo em que motivou o desenvolvimento de tecnologias capazes de dar suporte ao seu aprimoramento.

O paradigma da computação em nuvem vem ganhando cada vez mais força com a adesão de grandes empresas do ramo da computação, que vem realizando cada vez mais esforços no desenvolvimento de tecnologias a ele relacionadas.

2. Computação em Nuvem

O uso do termo C*loud Computing* ou *Computação em Nuvem* como também é conhecido no Brasil, vem se tornando cada vez mais freqüente, com a promessa de ser um paradigma que irá mudar a forma como os softwares são construídos e comercializados.

A idéia básica ao redor do computação em nuvem consiste em que tudo é serviço, uma espécie de *self-service* de tecnologias da informação. Muitos serviços de computação em nuvem *s*ão oferecidos pelo modelo conhecido como *Utility Computing*, o qual é definido como um pacote de recursos computacionais medidos e cobrados de forma semelhante aos serviços de utilidade pública, como eletricidade, água ou telefone. Este modo prevê um melhor investimento nos recursos de hardware, diminuindo a necessidade de grandes investimentos iniciais, possibilitando o aluguel de mais recursos à medida que forem necessários maiores recursos computacionais, podendo ainda alguns provedores de serviços se ajustarem dinamicamente às necessidades de demanda, ou seja, a possibilidade do cliente contratar recursos extras apenas para momentos de pico, evitando que recursos fiquem ociosos a maior parte do tempo.

Atualmente os serviços de *computação em nuvem* possuem três categorias bem definidas de acordo com os recursos e o modo que estes recursos são disponibilizados:

d) Infraestrutura como um Serviço (IaaS)
e) Plataforma como um Serviço (PaaS)
f) Software como um Serviço (SaaS)

A Infraestrutura como um serviço (IaaS), é a base dos serviços de computação em nuvem sendo a parte que se refere à disponibilização dos recursos de hardware, formados pelos servidores, armazenamento e processadores. Geralmente o fornecimento destes serviços se apóia em tecnologias como a de virtualização, onde o consumidor do serviço adquire uma máquina virtual operando sobre a estrutura de servidores do provedor, sendo a quantidade de processamento e armazenamento contratados disponibilizados através da configuração adequada de tais máquinas.

Plataforma como um Serviço (PaaS) é um tipo de serviço que procura prover toda a estrutura necessária para um ambiente de desenvolvimento. O alvo destes serviços são os desenvolvedores de software, tornando disponível uma plataforma computacional completa, agregando sistema operacional, linguagens de programação, bibliotecas, sistemas gerenciadores de banco de dados, etc. Neste serviço o ambiente de desenvolvimento em si está disponível através da nuvem, deste modo os desenvolvedores podem construir suas aplicações sem a necessidade de instalar qualquer ferramenta no seu computador e de distribuí-la de forma simplificada. Uma alternativa ao PaaS seria desenvolver as aplicações usando ferramentas de desenvolvimento para *desktops* e depois enviá-las para um provedor baseado em computação em nuvem .

Uma definição simplista para o modelo Software como um Serviço poderia ser a de software distribuído como um serviço hospedado e acessado através da internet. Porém se levarmos em conta apenas esta característica não teremos a especificação de nenhuma arquitetura – não faz referência a nenhuma tecnologia ou protocolos específicos – ou modelo de negócios específico, apenas define onde o código da aplicação reside, e como eles são distribuídos e acessados. Portanto, seguindo esta definição basta que a aplicação atenda ao seguinte critério: um fornecedor hospeda toda a lógica do programa e os dados, e provê acesso aos usuários finais através da internet por uma interface web. Porém para que uma aplicação tire total proveito deste modelo ela deve ter seu modelo de negócios, arquitetura e estrutura operacional visando suportar ao máximo a escalabilidade.

3. Google Application Engine

O Google Application Engine, ou App Engine como é comumente chamado é um conjunto de ferramentas e serviços disponibilizado pela empresa norte americana Google.

Trata-se de um modelo de PaaS, que diferentemente de seu conceito original, em que todo o ambiente responsável pelas etapas de desenvolvimento e publicação do software se dá através de ferramentas disponibilizadas via web, disponibiliza um ambiente desktop completo e de fácil configuração para esta finalidade.

Através delas o desenvolvedor tem a possibilidade de criar suas aplicações e enviá-las para os servidores que integram a nuvem do Google, podendo usufruir, além da plataforma de desenvolvimento, de sua infraestrutura de armazenamento, processamento e escalabilidade sem a necessidade de manter servidores, com os recursos podendo ser adquiridos da forma IaaS (Infraestrutura com um Serviço) de acordo com as cotas fornecidas pelo serviço à medida que se torne necessário maior poder de processamento, capacidade de tráfego ou armazenamento de dados para seu aplicativo.

Para este propósito o Google Application Engine possui ambientes com suporte às ultimas versões das linguagens Python e Java, fazendo uso de bibliotecas padrões além de fornecer um nome de domínio gratuito para acesso ao aplicativo enviado.

Para garantir o bom funcionamento das aplicações cada instância da aplicação fica restrita no seu próprio ambiente seguro e distinto tornando possível que ela seja distribuída entre os vários servidores sem que seja influenciada ou influencie o funcionamento de outras aplicações. Para que isto posso ocorrer cada aplicações está sujeira a uma série de limitações, entre elas a de escrita no sistema de arquivos, criação de threads e sockets.

3.1 Armazenamento de dados

Quando se fala em armazenamento de dados estamos tratando de algo vital para o Google Application Engine, pois é o que dá suporte para que este e outros serviços do Google possam ser oferecidos.

O sistema de armazenamento de dados do App Engine e Google em geral tem sua base apoiada sobre o sistema de arquivos chamado de *Google File System (GFS)*. O desenvolvimento de um novo sistema de arquivos distribuídos veio à tona em 2003 a partir da idéia de armazenar dados de forma escalável, confiável, com alto desempenho e disponibilidade mesmo em máquinas não confiáveis, devendo atender às necessidades de uso do Google de gerar e manter enormes quantidades de dados.

O BigTable é o sistema de gerenciamento de banco de dados que o Google desenvolveu apoiado sobre o *GFS* a fim de ter um banco de dados distribuído e de grande porte que atendesse ao seus propósitos. O desenvolvimento de um sistema sólido e eficiente

de banco de dados para rodar sobre a estrutura do Google propiciou o desenvolvimento de várias aplicações por ele fornecidas e o desenvolvimento do Google App Engine.

O App Engine não utiliza esquemas de modelagem de dados, ou seja, não são criadas tabelas específicas para cada tipo de entidade criada por uma aplicação, contudo utiliza um total de seis "Bigtables" – tabelas mantidas sob o sistema gerenciador de banco de dados do Google – para armazenar todas as entidades e índices, independente do tipo, de todas as aplicações. Tais tabelas estão dispostas de maneira que apenas uma seja usada para armazenar os dados em si e outras cinco para gerenciamento de índices.

Uma tabela armazena todas as entidades do serviço, mantendo os metadados, nomes e valores de propriedades e informações de índices personalizados, uma tabela automática de índices pelo tipo, duas tabelas para indexar propriedades e duas tabelas para armazenar índices personalizados.

3.2 Serviços do Google App Engine

O Google Application Engine conta com diversos serviços que facilitam a execução de determinadas operações por parte da aplicação. Para tanto, todas as linguagens de programação suportadas pelo App Engine oferecem APIs específicas para cada serviço em questão, descomplicando a execução de tarefas a eles relacionadas ou visando minimizar os efeitos das restrições impostas pela sandbox.

O serviço de contas do Google é responsável pela integração entre o aplicativo desenvolvido utilizando o App Engine com as contas do Google. Permite elaborar um sistema de *login* no aplicativo para que o usuário se autentique com o uso de sua conta do Google. Torna mais ágil o uso do aplicativo já que o usuário talvez não precise criar uma nova conta e economizando esforço na implementação de um sistema de contas específico para o aplicativo.

A partir do serviço de obtenção de URL a aplicação é capaz de acessar recursos na internet, como serviços da web e outras aplicações, emitindo solicitações *HTTP* ou *HTTPS* e recebendo respostas. Para tanto utiliza a mesma infraestrutura de alta velocidade do Google.

O App Engine fornece o serviço de cache de memória de alto desempenho capaz de ser compartilhado por diversas instâncias do aplicativo. Este tipo de serviço é extremamente útil para dados temporários que não necessitam ser persistidos no sistema de armazenamento de dados.

Os aplicativos também são capazes de enviar mensagens de e-mail facilmente fazendo uso da infraestrutura do Google através do serviço de mensagens fornecido pelo App Engine, manipular imagens através do serviço de manipulação de imagens ou enviar e receber mensagens instantâneas de qualquer serviço de mensagens instantâneas compatível com o protocolo XMPP, como o Google Talk.

Há ainda os serviços experimentais Task Queue e Blobstore. A partir do serviço de task queue as aplicações podem executar tarefas fora da requisição de um usuário, porém iniciada pela requisição. Se uma aplicação precisa executar algum trabalho de fundo, ela organiza este trabalho em pequenas tarefas e as coloca em uma ou mais filas. serviço Blobstore permite à aplicação servir objetos de dados, chamados de *Blobs*, que são muito maiores que os tamanhos permitidos para objetos no serviço de armazenamento de dados.

3.3 Cotas e faturamento

Os recursos do App Engine são medidos mediante dois tipos de cotas: uma cota faturável e uma cota fixa. As cotas faturáveis são os valores máximos definidos pelo

administrador da aplicação ou impostos pelos limites gratuitos do App Engine, ou seja, recursos extras que podem ser comprados pelo administrador do aplicativo. Cada aplicativo recebe uma cota faturável gratuitamente, podendo ser aumentada com a ativação do faturamento, definindo um orçamento diário e em seguida distribuindo o orçamento para as cotas através do painel de administração. O Valor cobrado será apenas pelos recursos que o aplicativo realmente usa acima dos estipulados pela cota gratuita.

Cotas fixas são valores máximos de recursos pré-definidos pelo Google App Engine a fim de garantir a integridade do sistema. Eles descrevem os limites da arquitetura nos quais todos os aplicativos devem ser executados e existem para garantir que um aplicativo não consuma demasiados recursos, a ponto de afetar o desempenho dos outros aplicativos executados no sistema.

O Google Application Engine registra o uso diário de recursos e os considera esgotados quando a quantidade utilizada atinge a cota estabelecida. Um dia é um período de 24 horas, começando à meia-noite pelo horário do pacífico (GMT -08:00). Ao começo de cada dia todas as medições referentes ao uso de recursos são reiniciadas com exceção do armazenamento de dados que sempre representam a atual quantidade de espaço utilizado pelo armazenamento de dados.

O App Engine limita o consumo de recursos no intervalo de um minuto utilizando um sistema de cotas fixas chamado de cotas por minuto, a fim de evitar que o aplicativo consuma toda sua cota em um período muito curto de tempo e impede que um aplicativo afete o funcionamento de outro ao monopolizar um determinado recurso.

Os aplicativos possuem uma série de recursos sujeitos às cotas com quantidades alocadas para um período de 24 horas. Quando todo o recurso é consumido, este fica indisponível até a renovação da cota, implicando que o aplicativo não funcionará corretamente até lá. Quando um recurso necessário para iniciar uma solicitação está esgotado o App Engine retorna um código de status HTTP 403, que significa acesso proibido e não executa a chamada.

3.4 Console de administração

O Console da administração do Google App Engine fornece acesso completo às configurações e às informações da aplicação. O acesso ao console de administração é feito através do seguinte endereço: *http://appengine.google.com/*.

Após acessar o endereço citado acima serão requisitados dados correspondentes a uma conta Google cadastrada. Caso o App Engine seja usado integrado com o Google Apps o endereço para acessar o console de administração sofre algumas modificações, se tornando o seguinte, onde *seu-dominio.com* é o domínio no Google Apps: http://appengine.google.com/a/*seu-dominio.com*.

Pelo uso do console de administração é possível criar um novo aplicativo, inspecionar dados de acesso, logs de erro e analisar tráfego, dentre outras funcionalidades.

Ao entrar no endereço do console de administração e obter acesso através da autenticação dos dados da conta Google é possível visualizar uma lista dos aplicativos existentes para a conta, de um máximo de 10 aplicativos, contendo o identificador, título e versão atualmente sendo disponibilizada do aplicativo. Caso o limite de 10 aplicativos ainda tenha sido atingido para a conta, há a possibilidade de criação de mais aplicativos. Ao se clicar no identificador do aplicativo é possível ter acesso ao console de administração do aplicativo, o qual está dividido em quatro macro-seções onde é possível visualizar estatísticas da aplicação, alterar configurações ou ativar o faturamento de recursos.

4. Google Application Engine e Java

O Google Application Engine disponibiliza seu ambiente de execução, servidor de desenvolvimento e APIs de serviços tanto para a linguagem Python quanto para a linguagem Java. Devido à sua popularidade e às facilidades de desenvolvimento, graças à completa integração com a IDE de desenvolvimento Eclipse, será abordada neste trabalho a integração entre o App Engine e a linguagem Java.

O ambiente Java do App Engine executa em uma JVM (Máquina Virtual Java) da versão 6 do Java, com suporte a servlets, biblioteca Java padrão, armazenamento de dados e serviço do App Engine. Apesar de executar os programas usando a versão 6 do Java, pode-se usar classes compiladas utilizando qualquer versão anterior. O suporte as bibliotecas padrões facilita o desenvolvimento de aplicações já que não requer grandes mudanças no desenvolvimento para o App Engine em relação aos servidores usuais. O App Engine também possui total integração com a IDE Eclipse através de um plugin contendo sua SDK (Kit de desenvolvimento local) completa e também do Google Web Toolkit. A SDK do Google App Engine possui suporte para a versão 5 do Java e posteriores, além de poder ser usado com os vários *frameworks* de desenvolvimento Java para web, como o *Struts* ou *Spring*.

O aplicativo é estruturado na forma padrão WAR (*Web Application Archive*) – semelhante ao padrão JAR do Java para aplicações desktop – utilizado para desenvolver aplicativos baseados na versão Java para servidores web conhecida como *Java 2 Enterprise Edition* ou *J2EE*. Há a separação do código e dos arquivos estáticos, além de um descritor de implementação na forma de um arquivo xml com nome *web.xml* juntamente com outros arquivos de configuração.

O armazenamento de dados do App Engine para Java suporta *JDO* (*Java Data Objects*) ou *JPA* (*Java Persistence API*), padrões Java para armazenamento de dados. Estes padrões para o armazenamento de dados são implementados pelo App Engine com o uso do DataNucleus Access Platform, a implementação de software livre escolhida pelo App Engine para dar suporte a estes padrões.

Todos os serviços disponibilizados pelo App Engine possuem sua implementação para Java, disponíveis através de APIs específicas. Além destes serviços o kit de desenvolvimento possui ferramentas para testes, um servidor local de desenvolvimento capaz de simular o ambiente real de execução do App Engine, integração com Google Web Toolkit e uma ferramenta capaz de realizar certas configurações no aplicativo chamada de *AppCfg*.

4.1 JDO

A sigla JDO significa *Java Data Objects*, ou algo como objetos de dados Java. Trata-se de um padrão Java para persistência de dados independente do sistema de banco de dados utilizado, ou seja, uma aplicação que utiliza este padrão para modelar suas entidades de dados pode utilizar como sistema de armazenamento o BigTable do Google ou migrar para algum banco de dados relacional, como o *MySQL*, sem que isso afete seu modelo de dados. Deste modo perde-se um pouco em eficiência de modelagem, mas se ganha muito em portabilidade.

Para que isto seja possível é necessário que haja uma implementação do padrão executando em nível de servidor JVM, responsável por mapear os modelos fisicamente ao sistema escolhido. O responsável por esta tarefa no App Engine é o *DataNucleus Access Platform* implementando a versão 2.3 do JDO.

A modelagem das entidades baseia-se nos chamados *POJOs* (*Plain Old Java Objects*), que nada mais são do que objetos simples dotados de atributos ou propriedades, e métodos de acesso (atribuição e obtenção) a elas, adicionados de anotações JDO para torná-los passíveis de persistência.

Os relacionamentos entre entidades podem ser de dois tipos: proprietários e não-proprietários. Relacionamentos proprietários definem que uma entidade pai controla o ciclo de vida da outra, como a sua criação e exclusão. Caso uma entidade pai proprietária seja excluída, as entidades filhas também o são em cascata. Por outro lado relacionamentos não proprietários não possuem este controle, ficando isto a cargo do aplicativo.

Todo o acesso ao serviço de armazenamento de dados usando JDO deve ser realizado por intermédio de uma instância da classe *PersistenceManager* obtida através de uma instância da classe *PersistenceManagerFactory*. Instâncias desta última classe são relativamente demoradas para inicializarem, o que encoraja o uso do padrão de projeto *Singleton*, onde uma única instância da classe serve a aplicação durante toda sua execução.

5. Google Web Toolkit

O Google Web Toolkit, ou GWT, é uma poderosa ferramenta para auxiliar o desenvolvimento web. Com ela é possível a criação de aplicações para web com o código executado pelo cliente sendo feito em Java e trabalhar comunicações assíncronas com o servidor web, conhecidas como AJAX.

Para que isto possa acontecer, a ferramenta trabalha com o código dividido em duas partes distintas, uma com código a ser executado pelo cliente e outra com código a ser executado pelo servidor através de chamadas assíncronas feitas pelo código no cliente. O código Java destinado a ser executado pelo cliente é automaticamente compilado pelo Google Web Toolkit para código JavaScript capaz de ser executado por navegadores, deixando a cargo do compilador questões de compatibilidade entre navegadores e de otimização.

Aplicativos baseados neste conceito são muito mais agradáveis ao uso pelo cliente por evitar o uso excessivo de solicitações e se aproximar da interação proporcionada por aplicativos para desktop, evitando freqüentes carregamentos das páginas do aplicativo, melhorando a experiência de uso, algo que é um dos principais problemas de aplicativos web. O próprio Google utiliza esta ferramenta em vários de seus aplicativos, como no seu aplicativo de e-mail *Gmail*, a rede social *Orkut* e o conjunto de ferramentas como processadores de texto e planilhas do *Google Docs*.

Esta ferramenta pode ser utilizada por qualquer aplicativo Java para web e possui integração com o Google Application Engine sendo distribuída juntamente com seu plug-in para Eclipse.

5.1 Chamadas RPC

É impossível de construir uma aplicação web robusta e que execute código apenas do lado do cliente. Por conta disso o GWT oferece um mecanismo capaz de executar código no lado do servidor, chamado de *RPC* (*Remote Procedure Calls*), ou chamadas de procedimento remoto, que são invocadas pelos mecanismos GWT através de solicitações assíncronas ao servidor.

Para se criar um serviço *RPC* é necessário definir uma interface para o serviço que estenda a interface *RemoteService* do GWT , definir uma interface assíncrona e implementar uma classe que estenda *RemoteServiceServlet* com o código efetivamente.

Para realizar uma chamada RPC no código do cliente, é necessário instanciar a interface do serviço através do método *GWT.create()* para posteriormente chamar seus métodos de acordo com a assinatura disponibilizada pela interface assíncrona.

5.2 Construindo aplicações

Uma aplicação que utiliza o GWT basicamente possui código a ser executado pelo cliente, código a ser executado pelo servidor, além de definições de módulos e arquivos estáticos.

As aplicações GWT são divididas em uma ou mais partes chamadas de módulos. Cada módulo deve ser definido na raiz do seu pacote através de um arquivo de configuração XML com o nome do módulo acrescido do sufixo ”*.gwt.xml*”.

6. Exemplo de aplicação

Com o objetivo de completar os estudos acerca das tecnologias apresentadas, foi desenvolvida uma aplicação capaz de trabalhar com alguns dos aspectos apresentados.

Trata-se de um sistema de votação desenvolvido para web com o auxílio da ferramenta Google web Toolkit, através do qual seja possível criar votações, votar em uma votação e retirar relatórios das votações. Tudo isto deve funcionar hospedado e executado sob a estrutura do Google Application Engine.

Para a realização do projeto foi utilizada a tecnologia Java para desenvolvimento de aplicações web através da versão 6 do seu kit de desenvolvimento *Java2EE* e versão 1.3.3 do kit de desenvolvimento do Google Web Toolkit. Como suporte foi utilizado a ferramenta Google Web Toolkit versão 2.03, o ambiente de desenvolvimento Eclipse e ferramentas de controle de versão.

Para atender às especificações propostas a aplicação foi divida em três módulos distintos, um para cada papel nelas descritos, responsáveis por todos os processos necessários para uma votação. Cada módulo é implementado com o uso das técnicas fornecidas pelo Google Web Toolkit, possuindo sua interface gráfica totalmente feita com o uso de *widgets* do GWT a fim de possibilitar uma experiência de uso agradável ao usuário, com respostas rápidas e acesso assíncrono ao servidor da aplicação, contornando a sensação incômoda gerada pelo uso de aplicativos que não fazem uso de tecnologia semelhante. Cada módulo compartilha com os demais o mesmo modelo de dados definido pela aplicação e é acessado por meio da página HTML que o hospeda.

A arquitetura da aplicação se baseia na mescla dos padrões de projeto MVC (*Model View Controller)* e *Mediator* adaptados às necessidades causadas pelas características de divisão de código entre cliente e servidor do Google Web Toolkit.

O módulo administrativo reúne as funções destinadas aos administradores do sistema. Para acessá-lo é necessário que um administrador da aplicação se autentique com sua conta do Google, podendo então ver a lista de votações criadas e criar, excluir, editar ou ver relatórios de votações.

O módulo de auditoria reúne as funções destinadas aos auditores de uma votação. Para acessá-lo é necessário que o auditor de uma votação se autentique com sua conta do Google, podendo então ver a lista de votações que audita, podendo ver seus relatórios.

O módulo de votação permite que o votante de uma votação vote através de uma interface gráfica simples que mostra as informações necessárias para o voto. Este módulo é implementado como os módulos anteriores, comunicando-se com o seu serviço RPC para obter os dados da votação referente ao votante ou enviar as informações de voto, além de verificar se estas informações são válidas.

7. Conclusão

O Google se destacou ao desenvolver o Google Application Engine como serviço pioneiro, que dispõe ao mesmo tempo de plataforma e de infraestrutura como serviços para a criação aplicações web.

A grande inovação, e ponto forte deste serviço está em por em prática um conjunto dos conceitos da computação em nuvem ao providenciar uma completa plataforma para desenvolvimento de aplicações, contando com kit de desenvolvimento e ambiente de execução libertando o desenvolvedor de maiores preocupações técnicas, ambos automaticamente escaláveis. Seu ponto chave está em a aplicação poder atender uma crescente demanda de solicitações e armazenamento.

Entretanto, a série de limites que seu ambiente de execução impõe impede que se faça uso de todo o potencial de escalabilidade que o App Engine é capaz de alcançar. A principal desvantagem está relacionada ao seu sistema de armazenamento de dados, que apesar de todo seu potencial possui uma série de restrições, como por exemplo, o tamanho máximo que uma entidade pode possuir. Para tentar contornar os inconvenientes destes limites o Google providencia alguns recursos na forma de serviços.

É importante ressaltar que por se tratar de uma ferramenta nova e em constante desenvolvimento, seu amadurecimento pode sanar as dificuldades apresentadas. Um exemplo disso é o desenvolvimento do App Engine para negócios, destinado a empresas, com previsão de implementação de serviços de banco de dados em SQL e conexões SSL em todos os domínios da aplicação.

O Google Web Toolkit mostrou-se uma ferramenta com enorme potencial para facilitar a criação de aplicações com interfaces ricas integradas ao Java. Esta ferramenta possibilita através da sua integração com a linguagem Java, que se desenvolvam aplicações que utilizam ao máximo os recursos que os navegadores disponibilizam para a criação de aplicações web modernas.

Referências

APACHE. *Java Data Objects (JDO)*. http://db.apache.org/jdo/ (acesso em Abril de 2010).

BARBOSA, Fernando P., e Andrea S. CHARAO. "Grid Computing e Cloud Computing – Uma Relação de Diferenças, Semelhanças, Aplicabilidade e Possibilidades de Cooperação entre os dois Paradigmas." *Programa de Pós-Graduação em Informática – Universidade Federal de Santa Maria.*

CHANG, Fay et al. "Bigtable: A Distributed Storage System for Structured Data." *OSDI*, 2006: 14.

CHONG, Frederick, e Gianpaolo CARRARO. "Architecture Strategies for Catching the Long Tail." *Microsoft Corporation.* Abril de 2006. http://msdn.microsoft.com/en-us/library/aa479069.aspx (acesso em Abril de 2010).

GOOGLE. "Guia do desenvolvedor." *Google App Engine.* 2010. http://code.google.com/intl/pt-BR/appengine/docs/ (acesso em Maio de 2010).

GOOGLE. "Guia do desenvolvedor." *Google Web Toolkit.* 2010. http://code.google.com/intl/pt-BR/webtoolkit/overview.html (acesso em Maio de 2010).

10.2 Anexo B – Código Fonte

```
package voting.client.admin;

import java.util.List;

import voting.client.BaseView;
import voting.client.widgets.ReportPanel;
import voting.client.widgets.TopPanel;
import voting.client.widgets.admin.AdminVotingListView;
import voting.client.widgets.admin.AdminMenu;
import voting.model.VotingModel;
import voting.model.VotingReport;

import com.google.gwt.core.client.GWT;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

public class Admin implements BaseView {

    private final AdminServiceAsync adminService = (AdminServiceAsync) GWT
    .create(AdminService.class);

    private VerticalPanel leftPanel = new VerticalPanel();
    private VerticalPanel mainPanel = new VerticalPanel();
    private VerticalPanel topPanel     = new VerticalPanel();
```

```
private VotingModel.Status filter;

public void onModuleLoad() {
        DockPanel dockMainPanel = new DockPanel();
        dockMainPanel.setStyleName("cw-DockPanel");

        VerticalPanel panel = new VerticalPanel();
        panel.add(mainPanel);
        panel.setHeight("100%");
        panel.setWidth("100%");
        panel.setCellWidth(this.topPanel, "100%");
        panel.setCellWidth(this.mainPanel, "100%");
        panel.setCellHeight(this.mainPanel, "100%");

        dockMainPanel.add(this.topPanel,DockPanel.NORTH);
        dockMainPanel.add(this.leftPanel,DockPanel.WEST);
        dockMainPanel.add(panel,DockPanel.EAST);
        dockMainPanel.setHeight("100%");
        dockMainPanel.setWidth("100%");

        dockMainPanel.setCellWidth(panel, "100%");
        dockMainPanel.setCellHeight(panel, "100%");

        topPanel.addStyleName("topPanel");
    leftPanel.addStyleName("leftPanel");
    mainPanel.addStyleName("mainPanel");
    mainPanel.setWidth("100%");
    mainPanel.setHeight("100%");
    topPanel.setWidth("100%");

    this.filter = VotingModel.Status.ALL;

    showLeft();
```

```
        showTop();

        RootPanel.get("maincontainer").add(dockMainPanel);
    }

    public void showLeft() {
        AdminMenu menu = new AdminMenu(this);
        this.leftPanel.add(menu);
    }

    public void showTop() {
        topPanel.clear();
        String title = "";
        switch (this.filter) {
            case FINISHED:
                title = "Encerradas";
                break;
            case RUNNING:
                title = "Em andamento";
                break;
            case WAITING:
                title = "Não iniciadas";
                break;
            case ALL:
                title = "Todas as votações";
        }

        TopPanel panel = new TopPanel(title,"<img src='images/admtop.png'
hspace='3'>");
        panel.addCommand("Adicionar", new ClickHandler() {
            public void onClick(final ClickEvent event) {
              add();
            }
```

```
                });

        topPanel.add(panel);
    }

    @Override
    public void showDefault() {
        showList(filter);
    }

    public void showList(VotingModel.Status filter) {
        this.filter = filter;
        final Admin own = this;

        showTop();

        adminService.getVotingList(filter,
                new AsyncCallback<List<VotingModel>>() {
                    public void onFailure(Throwable caught) {
                        Window.alert("Erro ao obter lista de
votações: "+caught.getMessage());
                    }

                    public void onSuccess(List<VotingModel> result) {
                        AdminVotingListView panel = new
AdminVotingListView(result,own);
                        setWidget(panel);
                    }
                });
    }

    public void setWidget(Widget w) {
        this.mainPanel.clear();
```

```
            this.mainPanel.add(w);
        }

        public void add() {
            AdminVotingPanel votingPanel = new AdminVotingPanel(this);
            setWidget(votingPanel);
        }

        public void edit(String key) {
            final Admin view = this;
            adminService.getVoting(key,
                    new AsyncCallback<VotingModel>() {
                        public void onFailure(Throwable caught) {
                            Window.alert(caught.getMessage());
                        }

                        public void onSuccess(VotingModel result) {
                            AdminVotingPanel votingPanel = new
AdminVotingPanel(result,view);
                            setWidget(votingPanel);
                        }
                    });
        }

        public void report(String key) {
            final Admin view = this;
            adminService.getVotingReport(key,
                    new AsyncCallback<VotingReport>() {
                        public void onFailure(Throwable caught) {
                            Window.alert(caught.getMessage());
                        }

                        public void onSuccess(VotingReport result) {
```

```
                                        ReportPanel reportPanel = new
ReportPanel(result,view);
                                        setWidget(reportPanel);
                                }
                        });
        }

        public void saveVoting(VotingModel votingModel) {
                final VotingModel.Status filter = this.filter;
                adminService.saveVoting(votingModel,
                                new AsyncCallback<Void>() {
                                        public void onFailure(Throwable caught) {
                                                Window.alert(caught.getMessage());
                                        }

                                        public void onSuccess(Void result) {
                                                Window.alert("Votação salva com
sucesso!");
                                                showList(filter);
                                        }
                                });
        }

        public void removeVoting(String key) {
                final VotingModel.Status filter = this.filter;
                adminService.removeVoting(key,
                                new AsyncCallback<Void>() {
                                        public void onFailure(Throwable caught) {
                                                Window.alert(caught.getMessage());
                                        }

                                        public void onSuccess(Void result) {
```

```
                                        Window.alert("Votação excluída com
sucesso!");
                                        showList(filter);

                                    }
                                });
        }

}
package voting.client.widgets.admin;

import voting.client.admin.Admin;
import voting.client.widgets.TitleCommandBar;
import voting.model.VotingModel;
import voting.model.VotingModel.Status;
import voting.util.VUtils;

import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

public class AdminListItem extends Composite {

        private final Admin view;
        private VotingModel votingModel;

        private VerticalPanel mainPanel = new VerticalPanel();
        private TitleCommandBar title;
        private FlexTable fields = new FlexTable();
```

```
public AdminListItem(VotingModel votingModel, Admin view) {
        this.view = view;
        this.votingModel = votingModel;

        this.mainPanel.setWidth("100%");
        fields.setWidth("100%");

        title = new TitleCommandBar(this.votingModel.getTitle(),"<img src='images/voting32.png' hspace='3'>");

        if (votingModel.getStatus() == Status.WAITING) {
            title.addCommand("editar", new ClickListener(){
                public void onClick( Widget sender ) {
                    onEdit();
                }
            });
        }

        if (votingModel.getStatus() != Status.RUNNING) {
            title.addCommand("remover", new ClickListener(){
                public void onClick( Widget sender ) {
                    if( Window.confirm( "Tem certeza que deseja remover esta votacao?") ){
                        onRemove();
                    }
                }
            });
        }

        if (votingModel.getStatus() != Status.WAITING) {
            title.addCommand("relatório", new ClickListener(){
                public void onClick( Widget sender ) {
```

```
                                onReport();
                        }
                });
        }

        mainPanel.add( title );
        mainPanel.add(fields);

        fields.getRowFormatter().setStyleName(0, "AdminListItem-header");
        mainPanel.setStyleName("AdminListItem");

        buildItem();
        initWidget( this.mainPanel );
    }

    protected void onRemove() {
        this.view.removeVoting(this.votingModel.getId());
    }

    protected void onEdit() {
        this.view.edit(this.votingModel.getId());
    }

    protected void onReport() {
        this.view.report(this.votingModel.getId());
    }

    public void buildItem() {
        addField("ID",votingModel.getId(),"");
        addField("Descrição",votingModel.getDescription(),"");

    addField("Status",VUtils.StatusToString(votingModel.getStatus()),"status"+Stri
ng.valueOf(votingModel.getStatus()));
```

```
	}

	protected void addField(String name, String value, String style) {
		int row = fields.getRowCount();
		Label fieldName = new Label( name );
		Label fieldValue = new Label( value );
		fieldName.setStyleName("AdminListItem-name");
		fieldValue.setStyleName("AdminListItem-value");
		fields.getCellFormatter().setWidth(row,1,"100%");
		fields.setWidget(row, 0, fieldName);
		fields.setWidget(row, 1, fieldValue);
		if (!style.isEmpty()) {
			fieldName.addStyleName(style);
			fieldValue.addStyleName(style);
		} else {
			int t = row%2;
			fieldName.addStyleName("AdminListItem-
row"+String.valueOf(t));
			fieldValue.addStyleName("AdminListItem-
row"+String.valueOf(t));
		}
	}

}
package voting.client.widgets.admin;

import voting.client.admin.Admin;
import voting.client.widgets.StatusTreeItem;
import voting.model.VotingModel;

import com.google.gwt.event.logical.shared.SelectionEvent;
import com.google.gwt.event.logical.shared.SelectionHandler;
import com.google.gwt.user.client.ui.Composite;
```

```
import com.google.gwt.user.client.ui.DecoratedStackPanel;
import com.google.gwt.user.client.ui.Tree;

@SuppressWarnings("unchecked")
public class AdminMenu extends Composite implements SelectionHandler {

	private final Admin view;
	private DecoratedStackPanel stackPanel = new DecoratedStackPanel();

	public AdminMenu(Admin view) {
		this.view = view;

		Tree mTree = new Tree();

		StatusTreeItem mPanelRoot = new StatusTreeItem("Todas as votações",VotingModel.Status.ALL,this.view);
		mTree.addItem(mPanelRoot);

		StatusTreeItem item1 = new StatusTreeItem("Não iniciadas",VotingModel.Status.WAITING,this.view);
		mPanelRoot.addItem(item1);

		StatusTreeItem item2 = new StatusTreeItem("Encerradas",VotingModel.Status.FINISHED,this.view);
		mPanelRoot.addItem(item2);

		StatusTreeItem item3 = new StatusTreeItem("Em andamento",VotingModel.Status.RUNNING,this.view);
		mPanelRoot.addItem(item3);

		mPanelRoot.setState(true);
		mTree.addSelectionHandler(this);
		stackPanel.add(mTree,"Votações",true);
```

```
            initWidget(stackPanel);

            mTree.setSelectedItem(mPanelRoot);
            mPanelRoot.onTreeItemSelected();
        }

        @Override
        public void onSelection(SelectionEvent event) {
                StatusTreeItem tree = (StatusTreeItem)event.getSelectedItem();
                tree.onTreeItemSelected();
        }
}
package voting.client.admin;

import java.util.List;

import voting.error.ValidationException;
import voting.error.VotingException;
import voting.model.VotingModel;
import voting.model.VotingReport;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;


/**
 * The client side stub for the RPC service.
 */
@RemoteServiceRelativePath("admin")
public interface AdminService extends RemoteService {
        void removeVoting(String key) throws VotingException, ValidationException;
        void saveVoting(VotingModel v) throws VotingException, ValidationException;
```

```
	VotingModel getVoting(String key) throws VotingException,
ValidationException;
	VotingReport getVotingReport(String key) throws VotingException,
ValidationException;

	List<VotingModel> getVotingList(VotingModel.Status filter) throws
VotingException;
}
package voting.client.admin;

import java.util.List;

import voting.model.VotingModel;
import voting.model.VotingReport;

import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 * The async counterpart of <code>AdminService</code>.
 */
public interface AdminServiceAsync {
	void getVotingList(VotingModel.Status
filter,AsyncCallback<List<VotingModel>> callback);

	void removeVoting(String key, AsyncCallback<Void> callback);

	void getVoting(String key, AsyncCallback<VotingModel> callback);

	void saveVoting(VotingModel v, AsyncCallback<Void> callback);

	void getVotingReport(String key, AsyncCallback<VotingReport> callback);
}
package voting.server.admin;
```

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.servlet.ServletException;

import voting.client.admin.AdminService;
import voting.dao.PMF;
import voting.dao.VotingDAO;
import voting.error.ErrorMsg;
import voting.error.ValidationException;
import voting.error.VotingException;
import voting.model.Voter;
import voting.model.VotingModel;
import voting.model.VotingReport;
import voting.model.VotingModel.Status;
import voting.model.VotingModel.Type;
import voting.server.MailSender;

import com.google.appengine.repackaged.org.apache.commons.logging.Log;
import com.google.appengine.repackaged.org.apache.commons.logging.LogFactory;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

/**
 * The server side implementation of the RPC service.
 */
public class AdminServiceImpl extends RemoteServiceServlet implements
                AdminService {

        /**
         * Serialization ID
         */
```

```
	private static final long serialVersionUID = 14351417745501704 68L;

	/**
	 * Log channel
	 */
	private static Log _log = LogFactory.getLog(AdminServiceImpl.class);


	@Override
	public void init() throws ServletException
	{
		super.init();
	}

	protected boolean validateSave(VotingModel v) throws ValidationException  {
		if (v.getTitle().trim().equals("")) {
			throw new ValidationException("Título não preenchido.");
		}
		if (v.getDescription().trim().equals("")) {
			throw new ValidationException("Descrição não preenchida.");
		}
		if (v.getStart().compareTo(new Date()) <= 0) {
			throw new ValidationException("A data e hora de ínicio deve ser
maior que a atual.");
		}
		if (v.getStart().compareTo(v.getEnd()) >= 0) {
			throw new ValidationException("A data de início deve ser maior
que a do fim da votação.");
		}
		if (v.getType() == Type.OPTION) {
			if (v.getOptions().size() == 0) {
				throw new ValidationException("É necessária ao menos
uma opção.");
```

```
                }
            }
            if (v.getVoters().size() == 0) {
                throw new ValidationException("É necessário ao menos um
eleitor.");
            }

            return true;
        }

        protected boolean validateStatus(VotingModel v) throws ValidationException  {
            if (v.getStatus() == Status.RUNNING) {
                throw new ValidationException("A votação já foi iniciada.");
            }
            if (v.getStatus() == Status.FINISHED) {
                throw new ValidationException("A votação já foi finalizada.");
            }

            return true;
        }

        protected boolean validateStatusRemove(VotingModel v) throws
ValidationException  {
            if (v.getStatus() == Status.RUNNING) {
                throw new ValidationException("Não é possível remover uma
votação em andamento.");
            }

            return true;
        }

        protected boolean validateReport(VotingReport v) throws ValidationException
{
```

```
            if (v.getStatus() == Status.WAITING) {
                throw new ValidationException("A votação não foi iniciada.");
            }

            return true;
        }

        public VotingModel getVoting(String key) throws VotingException,
ValidationException {
            VotingModel result = null;
            VotingDAO dao = new VotingDAO(PMF.get());

            try {
                result = dao.loadVoting(key);
            }
            catch (Exception e) {
                _log.error(e.getMessage(), e);
                throw new VotingException(ErrorMsg.DATABASE);
      }

            validateStatus(result);

            return result;
        }

        public void removeVoting(String key) throws VotingException,
ValidationException {
            VotingDAO dao = new VotingDAO(PMF.get());
            VotingModel v;

            try {
                v = dao.loadVoting(key);
            }
```

```
            catch (Exception e) {
                _log.error(e.getMessage(), e);
                throw new VotingException(ErrorMsg.DATABASE);
        }

            validateStatusRemove(v);

            try {
                dao.deleteVoting(v);
            }
            catch (Exception e) {
                _log.error(e.getMessage(), e);
                throw new VotingException(ErrorMsg.DATABASE);
        }
          }

        public void saveVoting(VotingModel v) throws VotingException,
ValidationException {
            if (validateStatus(v) && validateSave(v)) {
                VotingDAO dao = new VotingDAO(PMF.get());

                ArrayList<Voter> newVoters = new ArrayList<Voter>();
                for (int i=0; i < v.getVoters().size(); i++) {
                    Voter aux = v.getVoters().get(i);
                    if (aux.getId() == null) {
                        newVoters.add(aux);
                    }
                }

                try {
                    v = dao.storeVoting(v);
                    MailSender mail = new MailSender();
                mail.sendAddMessage(newVoters);
```

```
                }
                catch (Exception e) {
                        _log.error(e.getMessage(), e);
                        throw new VotingException(ErrorMsg.DATABASE);
            }
              }
        }

        public List<VotingModel> getVotingList(VotingModel.Status filter) throws VotingException {
                List<VotingModel> result = new ArrayList<VotingModel>();

                VotingDAO dao = new VotingDAO(PMF.get());

                try {
                        List<VotingModel> list = dao.getVotings(filter);

                        for (int i=0;i < list.size();i++) {
                        VotingModel v = list.get(i);
                        result.add(v);
                }
                }
                catch (Exception e) {
                        _log.error(e.getMessage(), e);
                        throw new VotingException(ErrorMsg.DATABASE);
      }

                return result;
        }

        public VotingReport getVotingReport(String key) throws VotingException, ValidationException {
                VotingReport result = null;
```

```
                VotingDAO dao = new VotingDAO(PMF.get());

                try {
                        result = dao.getVotingReport(key);
                }
                catch (Exception e) {
                        _log.error(e.getMessage(), e);
                        throw new VotingException(ErrorMsg.DATABASE);
        }

                validateReport(result);

                return result;
          }

}package voting.client.widgets.admin;

import java.util.List;

import voting.client.admin.Admin;
import voting.model.VotingModel;

import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.VerticalPanel;

public class AdminVotingListView extends Composite {

        private final Admin view;
        private List<VotingModel> votingModels;

        private VerticalPanel mainPanel = new VerticalPanel();

        public AdminVotingListView(List<VotingModel> votingModels, Admin view) {
```

```
		this.view = view;
		this.votingModels = votingModels;

		this.mainPanel.setWidth("100%");

		buildList();

		initWidget( this.mainPanel );
	}

	public void buildList() {
		for (int i=0; i < votingModels.size(); i++) {
			AdminListItem item = new
AdminListItem(votingModels.get(i),this.view);
			mainPanel.add(item);
		}
	}

}
package voting.client.admin;

import voting.client.widgets.AuditorListView;
import voting.client.widgets.OptionListView;
import voting.client.widgets.TitlePanel;
import voting.client.widgets.VoterListView;
import voting.client.widgets.dialogs.DateTimeDialog;
import voting.model.VotingModel;
import voting.model.VotingModel.Type;

import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
```

```
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RadioButton;
import com.google.gwt.user.client.ui.TextArea;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

@SuppressWarnings("deprecation")
public class AdminVotingPanel extends Composite {

	/*
	 * Data Transfer objects que encapsulam os campos da votacao
	 */
	private VotingModel votingModel;

	/*
	 * Referencia a classe principal
	 */
	private Admin view;

	/*
	 * Objetos da interface grafica que representam a votacao
	 */
	private Label lbId = new Label();

	private TextBox edTitle = new TextBox();
	private TextArea edDescription = new TextArea();

	private RadioButton rbType1 = new RadioButton("tipo", "Enquete");
	private RadioButton rbType2 = new RadioButton("tipo", "Eleição");
```

```
OptionListView optionsPanel = new OptionListView("Opções");
VoterListView votersPanel = new VoterListView("Eleitores");
AuditorListView auditorsPanel = new AuditorListView("Auditores");

private Label lbStart = new Label();
private Label lbEnd = new Label();


public AdminVotingPanel(VotingModel votingModel, Admin view) {
        this.votingModel = votingModel;
        this.view = view;
        init();
}

public AdminVotingPanel(Admin view) {
        this.votingModel = new VotingModel();
        this.view = view;
        init();
}

protected void init() {
        //Container principal que ser[illegible] retornado
        VerticalPanel panel = new VerticalPanel();
        panel.setWidth("100%");
        initWidget(panel);

        TitlePanel tpId = new TitlePanel("Id");
        TitlePanel tpTitle = new TitlePanel("Título");
        TitlePanel tpDescription = new TitlePanel("Descrição");
        TitlePanel tpType = new TitlePanel("Tipo");
```

```
        //Container ID
        tpId.addWidget(lbId);
        panel.add(tpId);

        //Container Titulo
        tpTitle.addWidget(edTitle);
        panel.add(tpTitle);

        //Container Descricao
        tpDescription.addWidget(edDescription);
        panel.add(tpDescription);

        //Container Tipo
        tpType.addWidget(rbType1);
        tpType.addWidget(rbType2);
        panel.add(tpType);

        //Datas
        panel.add(createDateStartPanel());
        panel.add(createDateEndPanel());

        //Lista de opcoes e eleitores
        panel.add(optionsPanel);
    panel.add(votersPanel);
    panel.add(auditorsPanel);

        // Botoes
    panel.add(createButtons());

    showProperties();
}

protected Widget createButtons() {
```

```
    HorizontalPanel panel   = new HorizontalPanel();

    Button okButton = new Button("Salvar",new ClickHandler() {
                public void onClick(ClickEvent event) {
                        onSave();
                }
        });
    Button cancelButton = new Button("Cancelar",new ClickHandler() {
                public void onClick(ClickEvent event) {
                        onCancel();
                }
        });

    panel.setStyleName("AdminVotingPanel-buttons");

    panel.add(okButton);
    panel.add(cancelButton);

    return panel;
}

protected void onSave() {

        this.votingModel.setTitle(edTitle.getText());
        this.votingModel.setDescription(edDescription.getText());
        this.votingModel.setType(this.getType());

        if (this.validate()) {
                view.saveVoting(this.votingModel);
        }
}

protected void onCancel() {
```

```
        view.showDefault();
}

public void showProperties() {
        lbId.setText(votingModel.getId());
        edTitle.setText(votingModel.getTitle());
        edDescription.setText(votingModel.getDescription());
        setType(votingModel.getType());

        showDates();

        optionsPanel.setList(votingModel.getOptions());
        votersPanel.setList(votingModel.getVoters());
        auditorsPanel.setList(votingModel.getAuditors());
}

public void showDates() {
        if (votingModel.getStart() != null) {
                lbStart.setText(votingModel.getStart().toLocaleString());
        }
        if (votingModel.getEnd() != null) {
                lbEnd.setText(votingModel.getEnd().toLocaleString());
        }
}


/*
 * TIPO
 */
protected VotingModel.Type getType() {
        VotingModel.Type result = VotingModel.Type.OPTION;

        if (rbType1.getValue()) {
```

```
                result = VotingModel.Type.OPTION;
        }
        else if (rbType2.getValue()) {
                result = VotingModel.Type.NOMINEE;
        }

        return result;
}
protected void setType(VotingModel.Type type) {
        switch (type) {
        case OPTION:
                rbType1.setValue(true);
                break;
        case NOMINEE:
                rbType2.setValue(true);
                break;
        default:
                rbType1.setValue(true);
                break;
        }
}

/*
 * DATA INICIO
 */
protected Widget createDateStartPanel() {
        TitlePanel tpStart = new TitlePanel("Início");

        HorizontalPanel panel       = new HorizontalPanel();
        panel.add(lbStart);

        final AdminVotingPanel view = this;
```

```
		tpStart.addCommand("Modificar", new ClickListener(){
			public void onClick( Widget sender ) {
				new DateTimeDialog("Data de
inicio",votingModel.getStart(),view);
			}
		});

		tpStart.addWidget(panel);
		return tpStart;
	}


	/*
	 * DATA FIM
	 */
	protected Widget createDateEndPanel() {
		TitlePanel tpEnd = new TitlePanel("Fim");

		HorizontalPanel panel	= new HorizontalPanel();
		panel.add(lbEnd);

		final AdminVotingPanel view = this;

		tpEnd.addCommand("Modificar", new ClickListener(){
			public void onClick( Widget sender ) {
				new DateTimeDialog("Data de
fim",votingModel.getEnd(),view);
			}
		});

		tpEnd.addWidget(panel);
		return tpEnd;
	}
```

```
	/*
	 * VALIDACAO DOS DADOS
	 */
	protected boolean validate() {
		if (votingModel.getTitle().trim().equals("")) {
			Window.alert("Preencha o título");
			this.edTitle.setFocus(true);
			return false;
		}
		if (votingModel.getDescription().trim().equals("")) {
			Window.alert("Preencha a descrição");
			this.edDescription.setFocus(true);
			return false;
		}
		if (votingModel.getStart().compareTo(votingModel.getEnd()) >= 0) {
			Window.alert("A data de ínicio da votação tem que ser menor
que a do fim");
			return false;
		}
		if (votingModel.getType() == Type.OPTION) {
			if (votingModel.getOptions().size() == 0) {
				Window.alert("Deve haver ao menos uma opção");
				return false;
			}
		}
		if (votingModel.getVoters().size() == 0) {
			Window.alert("Deve haver ao menos um eleitor");
			return false;
		}

		return true;
	}
```

```
}
package voting.client.auditor;

import java.util.List;

import voting.client.BaseView;
import voting.client.widgets.ReportPanel;
import voting.client.widgets.TopPanel;
import voting.client.widgets.auditor.AuditorMenu;
import voting.client.widgets.auditor.AuditorVotingListView;
import voting.model.VotingModel;
import voting.model.VotingReport;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

public class Auditor implements BaseView {

	private final AuditorServiceAsync auditorService = (AuditorServiceAsync) GWT
	.create(AuditorService.class);

	private VerticalPanel leftPanel = new VerticalPanel();
	private VerticalPanel mainPanel = new VerticalPanel();
	private VerticalPanel topPanel  = new VerticalPanel();
	private TopPanel titlePanel = new TopPanel("","<img src='images/auditortop.png' hspace='3'>");
```

```
private VotingModel.Status filter;

public void onModuleLoad() {
        DockPanel dockMainPanel = new DockPanel();
        dockMainPanel.setStyleName("cw-DockPanel");

        VerticalPanel panel = new VerticalPanel();
        //panel.add(topPanel);
        panel.add(mainPanel);
        panel.setHeight("100%");
        panel.setWidth("100%");
        panel.setCellWidth(this.topPanel, "100%");
        panel.setCellWidth(this.mainPanel, "100%");
        panel.setCellHeight(this.mainPanel, "100%");

        dockMainPanel.add(this.topPanel,DockPanel.NORTH);
        dockMainPanel.add(this.leftPanel,DockPanel.WEST);
        dockMainPanel.add(panel,DockPanel.EAST);
        dockMainPanel.setHeight("100%");
        dockMainPanel.setWidth("100%");

        dockMainPanel.setCellWidth(panel, "100%");
        dockMainPanel.setCellHeight(panel, "100%");

        topPanel.addStyleName("topPanel");
    leftPanel.addStyleName("leftPanel");
    mainPanel.addStyleName("mainPanel");
    mainPanel.setWidth("100%");
    mainPanel.setHeight("100%");
    topPanel.setWidth("100%");

    this.filter = VotingModel.Status.ALL;
```

```
    showLeft();
    showTop();

    RootPanel.get("maincontainer").add(dockMainPanel);

    this.getUser();
}

public void showLeft() {
        AuditorMenu menu = new AuditorMenu(this);
        this.leftPanel.add(menu);
}

public void showTop() {
        topPanel.clear();
    topPanel.add(titlePanel);
}

@Override
public void showDefault() {
        showList(filter);
}

public void getUser() {
        auditorService.getUser(
                        new AsyncCallback<String>() {
                                public void onFailure(Throwable caught) {
                                        Window.alert(caught.getMessage());
                                }

                                public void onSuccess(String result) {
                                        titlePanel.setText(result);
```

```
                }
            });
    }

    public void showList(VotingModel.Status filter) {
        this.filter = filter;
        final Auditor own = this;

        auditorService.getVotingList(filter,
                new AsyncCallback<List<VotingModel>>() {
                    public void onFailure(Throwable caught) {
                        Window.alert("Erro ao obter lista de
votações: "+caught.getMessage());
                    }

                    public void onSuccess(List<VotingModel> result) {
                        AuditorVotingListView panel = new
AuditorVotingListView(result,own);
                        setWidget(panel);
                    }
                });
    }

    public void setWidget(Widget w) {
        this.mainPanel.clear();
        this.mainPanel.add(w);
    }

    public void report(String key) {
        final Auditor view = this;
        auditorService.getVotingReport(key,
                new AsyncCallback<VotingReport>() {
                    public void onFailure(Throwable caught) {
```

```
                                        Window.alert(caught.getMessage());
                                    }

                                    public void onSuccess(VotingReport result) {
                                        ReportPanel reportPanel = new
ReportPanel(result,view);
                                        setWidget(reportPanel);
                                    }
                                });
        }

}
package voting.client.widgets.dialogs;

import voting.client.widgets.AuditorListView;

public class AuditorDialog extends ObjectDialogBox {

        private AuditorListView view;
        private String email;
        private DialogMode mode;

        public AuditorDialog(AuditorListView view) {
                super("<img src='images/voters24.png' hspace='3'>Eleitor");
                this.email = "";
                this.view = view;
                this.mode = DialogMode.ADD;
                init();
        }

        private void init() {
                addField( "E-mail", email );
                addButtons();
```

```
        }

        public void onSubmit(){
                email = getField(0);
                if (!email.equals("")) {
                        if (this.mode == DialogMode.ADD) {
                                view.add(email);
                        }
                }
        }

}
package voting.client.widgets.auditor;

import voting.client.auditor.Auditor;
import voting.client.widgets.TitleCommandBar;
import voting.model.VotingModel;
import voting.model.VotingModel.Status;
import voting.util.VUtils;

import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

public class AuditorListItem extends Composite {

        private final Auditor view;
        private VotingModel votingModel;

        private VerticalPanel mainPanel = new VerticalPanel();
```

```
	private TitleCommandBar title;
	private FlexTable fields = new FlexTable();

	public AuditorListItem(VotingModel votingModel, Auditor view) {
		this.view = view;
		this.votingModel = votingModel;

		this.mainPanel.setWidth("100%");
		fields.setWidth("100%");

		title = new TitleCommandBar(this.votingModel.getTitle(),"<img
src='images/voting32.png' hspace='3'>");

		if (votingModel.getStatus() != Status.WAITING) {
			title.addCommand("relatório", new ClickListener(){
				public void onClick( Widget sender ) {
					onReport();
				}
			});
		}

		mainPanel.add( title );
		mainPanel.add(fields);

		fields.getRowFormatter().setStyleName(0, "AdminListItem-header");
		mainPanel.setStyleName("AdminListItem");

		buildItem();
		initWidget( this.mainPanel );
	}

	protected void onReport() {
		this.view.report(this.votingModel.getId());
```

```
	}

	public void buildItem() {
		addField("ID",votingModel.getId(),"");
		addField("Descrição",votingModel.getDescription(),"");

	addField("Status",VUtils.StatusToString(votingModel.getStatus()),"status"+Stri
ng.valueOf(votingModel.getStatus()));
	}

	protected void addField(String name, String value, String style) {
		int row = fields.getRowCount();
		Label fieldName = new Label( name );
		Label fieldValue = new Label( value );
		fieldName.setStyleName("AdminListItem-name");
		fieldValue.setStyleName("AdminListItem-value");
		fields.getCellFormatter().setWidth(row,1,"100%");
		fields.setWidget(row, 0, fieldName);
		fields.setWidget(row, 1, fieldValue);
		if (!style.isEmpty()) {
			fieldName.addStyleName(style);
			fieldValue.addStyleName(style);
		} else {
			int t = row%2;
			fieldName.addStyleName("AdminListItem-
row"+String.valueOf(t));
			fieldValue.addStyleName("AdminListItem-
row"+String.valueOf(t));
		}
	}

}
package voting.client.widgets;
```

```
import java.util.List;

import voting.client.widgets.dialogs.AuditorDialog;

public class AuditorListView extends ListView {

	private List<String> list;

	public AuditorListView(String titleValue) {
		super(titleValue, "<img src='images/voters24.png'>", "Adicionar");
	}

	public void setList(List<String> list) {
		this.list = list;
		this.refresh();
	}

	public void refresh() {
		clear();

		int row = 0;
		for( java.util.Iterator<String> it = this.list.iterator(); it.hasNext();){
			String auditor = (String)it.next();
			addField(row,auditor);
			addRemoveField(row,"excluir");
			row++;
		}
	}

	public void add(String email) {
		this.list.add(email);
		refresh();
```

```
        }

        protected void onClear() {
                addColumn( "E-mail" );
                addColumn( "" );
        }

        protected void onAdd() {
                new AuditorDialog(this);
        }

        protected void onRemove(int index) {
                list.remove(index);
                refresh();
        }

}
package voting.client.widgets.auditor;

import voting.client.auditor.Auditor;
import voting.client.widgets.StatusTreeItem;
import voting.model.VotingModel;

import com.google.gwt.event.logical.shared.SelectionEvent;
import com.google.gwt.event.logical.shared.SelectionHandler;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.DecoratedStackPanel;
import com.google.gwt.user.client.ui.Tree;

@SuppressWarnings("unchecked")
public class AuditorMenu extends Composite implements SelectionHandler {
```

```
	private final Auditor view;
	private DecoratedStackPanel stackPanel = new DecoratedStackPanel();

	public AuditorMenu(Auditor view) {
		this.view = view;

		Tree mTree = new Tree();

		StatusTreeItem mPanelRoot = new StatusTreeItem("Todas as votações",VotingModel.Status.ALL,this.view);
		mTree.addItem(mPanelRoot);

	    StatusTreeItem item1 = new StatusTreeItem("Não iniciadas",VotingModel.Status.WAITING,this.view);
	    mPanelRoot.addItem(item1);

	    StatusTreeItem item2 = new StatusTreeItem("Encerradas",VotingModel.Status.FINISHED,this.view);
	    mPanelRoot.addItem(item2);

	    StatusTreeItem item3 = new StatusTreeItem("Em andamento",VotingModel.Status.RUNNING,this.view);
	    mPanelRoot.addItem(item3);

	    mPanelRoot.setState(true);
	    mTree.addSelectionHandler(this);
	    stackPanel.add(mTree,"Votações",true);

	    initWidget(stackPanel);

	    mTree.setSelectedItem(mPanelRoot);
	    mPanelRoot.onTreeItemSelected();
	}
```

```
		@Override
		public void onSelection(SelectionEvent event) {
			StatusTreeItem tree = (StatusTreeItem)event.getSelectedItem();
			tree.onTreeItemSelected();
		}

}
package voting.client.auditor;

import java.util.List;

import voting.error.ValidationException;
import voting.error.VotingException;
import voting.model.VotingModel;
import voting.model.VotingReport;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("auditor")
public interface AuditorService extends RemoteService {

		List<VotingModel> getVotingList(VotingModel.Status filter) throws
VotingException;
		VotingReport getVotingReport(String key) throws VotingException,
ValidationException;
		String getUser() throws VotingException, ValidationException;

}
package voting.client.auditor;

import java.util.List;
```

```
import voting.model.VotingModel;
import voting.model.VotingReport;
import voting.model.VotingModel.Status;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface AuditorServiceAsync {

	void getVotingList(Status filter, AsyncCallback<List<VotingModel>> callback);

	void getVotingReport(String key, AsyncCallback<VotingReport> callback);

	void getUser(AsyncCallback<String> callback);

}
package voting.server.auditor;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;

import voting.client.auditor.AuditorService;
import voting.dao.PMF;
import voting.dao.VotingDAO;
import voting.error.ErrorMsg;
import voting.error.ValidationException;
import voting.error.VotingException;
import voting.model.VotingModel;
import voting.model.VotingReport;
import voting.model.VotingModel.Status;
import voting.server.admin.AdminServiceImpl;
```

```
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;
import com.google.appengine.repackaged.org.apache.commons.logging.Log;
import com.google.appengine.repackaged.org.apache.commons.logging.LogFactory;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class AuditorServiceImpl extends RemoteServiceServlet implements
		AuditorService {

	/**
	 *
	 */
	private static final long serialVersionUID = 8266322485375751516L;

	/**
	 * Log channel
	 */
	private static Log _log = LogFactory.getLog(AdminServiceImpl.class);

	@Override
	public void init() throws ServletException
	{
		super.init();
	}

	protected boolean validateReport(VotingModel v, String email) throws
ValidationException  {
		if (v.getStatus() == Status.WAITING) {
			throw new ValidationException("A votação não foi iniciada.");
		}
```

```
                boolean isAuditor = false;
                for (int i=0; i < v.getAuditors().size(); i++) {
                        if (v.getAuditors().get(i).equals(email)) {
                                isAuditor = true;
                                break;
                        }
                }
                if (!isAuditor) {
                        throw new ValidationException("Você não é auditor desta votação.");
                }

                return true;
        }

        @Override
        public List<VotingModel> getVotingList(Status filter) throws VotingException {
                List<VotingModel> result = new ArrayList<VotingModel>();

                UserService userService = UserServiceFactory.getUserService();
                User user = userService.getCurrentUser();
                if (user != null) {

                        VotingDAO dao = new VotingDAO(PMF.get());

                        try {
                                List<VotingModel> list = dao.getAuditorVotings(user.getEmail(),filter);

                                for (int i=0;i < list.size();i++) {
                                VotingModel v = list.get(i);
                                result.add(v);
                        }
```

```
                }
                catch (Exception e) {
                        _log.error(e.getMessage(), e);
                        throw new VotingException(ErrorMsg.DATABASE);
            }
              } else {
                        throw new VotingException(ErrorMsg.NOTLOGGEDIN);
              }

              return result;
        }

        @Override
        public VotingReport getVotingReport(String key) throws
VotingException,ValidationException {
              VotingReport result = null;
              VotingModel v;

              UserService userService = UserServiceFactory.getUserService();
              User user = userService.getCurrentUser();
              if (user != null) {
                        VotingDAO dao = new VotingDAO(PMF.get());

                        try {
                                v = dao.loadVoting(key);
                        }
                        catch (Exception e) {
                                _log.error(e.getMessage(), e);
                                throw new VotingException(ErrorMsg.DATABASE);
              }

                        validateReport(v, user.getEmail());
```

```
                try {
                        result = dao.getVotingReport(key);
                }
                catch (Exception e) {
                        _log.error(e.getMessage(), e);
                        throw new VotingException(ErrorMsg.DATABASE);
            }
              } else {
                        throw new VotingException(ErrorMsg.NOTLOGGEDIN);
              }

              return result;
        }

        @Override
        public String getUser() throws VotingException, ValidationException {
                String result = "";
                UserService userService = UserServiceFactory.getUserService();
                User user = userService.getCurrentUser();
                result = user.getNickname()+" - "+user.getEmail();
                return result;
        }

}
package voting.client.widgets.auditor;

import java.util.List;

import voting.client.auditor.Auditor;
import voting.model.VotingModel;

import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.VerticalPanel;
```

```
public class AuditorVotingListView extends Composite {

	private final Auditor view;
	private List<VotingModel> votingModels;

	private VerticalPanel mainPanel = new VerticalPanel();

	public AuditorVotingListView(List<VotingModel> votingModels, Auditor view) {
		this.view = view;
		this.votingModels = votingModels;

		this.mainPanel.setWidth("100%");

		buildList();

		initWidget( this.mainPanel );
	}

	public void buildList() {
		for (int i=0; i < votingModels.size(); i++) {
			AuditorListItem item = new
AuditorListItem(votingModels.get(i),this.view);
			mainPanel.add(item);
		}
	}

}
package voting.client.widgets;

import voting.client.BaseView;

import com.google.gwt.user.client.ui.TreeItem;
```

```
public class BaseTreeItem extends TreeItem {

	private final BaseView view;

	public BaseTreeItem(String html, BaseView view) {
		super(html);
		this.view = view;
	}

	public BaseView getView() {
		return view;
	}

	public boolean isSelected(){
		return getTree().getSelectedItem() == this;
	}

	public void onTreeItemSelected(){}
	public void onTreeItemStateChanged(){}
	public void onRefresh(){}
}
package voting.client.widgets.dialogs;

import java.util.Date;

import voting.client.admin.AdminVotingPanel;

import com.google.gwt.gen2.picker.client.TimePicker;
import com.google.gwt.i18n.client.DateTimeFormat;
import com.google.gwt.user.datepicker.client.DatePicker;

@SuppressWarnings("deprecation")
```

```
public class DateTimeDialog extends ObjectDialogBox {

	private AdminVotingPanel view;
	private Date date;

	private DatePicker dp = new DatePicker();
	final TimePicker tp = new TimePicker(new Date(), null,
		DateTimeFormat.getFormat("HH"),	DateTimeFormat.getFormat("mm"),
null);


	public DateTimeDialog(String string,Date date,AdminVotingPanel view) {
		super("<img src='images/clock24.png' hspace='3'>"+string);
		this.date = date;
		this.view = view;
		init();
	}

	private void init() {
		setDateTime();
		addField( "Data", dp );
		addField( "Hora", tp );
		addButtons();
	}

	protected void setDateTime() {
		if (date != null) {
			dp.setValue(date);
			//tp.setTime(date.getHours(), date.getMinutes());
			tp.setDateTime(date);
		} else {
			//tp.setTime(date.getHours(), date.getMinutes());
			tp.setDateTime(new Date());
```

```
                        dp.setValue(dp.getLastDate());
                }
        }

        protected void getDateTime() {
                date.setDate(dp.getValue().getDate());
                date.setHours(tp.getDateTime().getHours());
                date.setMinutes(tp.getDateTime().getMinutes());
        }

        public void onSubmit(){
                getDateTime();
                view.showDates();
        }

}
package voting.error;

public enum ErrorMsg {
        DEFAULT(0, "Ocorreu um erro."),
        DATABASE(1, "Ocorreu um erro no banco de dados."),
        NOTLOGGEDIN(2, "Não há usuário autenticado no sistema."),

        VOTING_FINISHED(1001, "A votação já foi encerrada."),
        VOTING_WAITING(1002, "A votação ainda não foi iniciada.");

        private final int code;
        private final String description;

        private ErrorMsg(int code, String description) {
                this.code = code;
                this.description = description;
        }
```

```
	public String getDescription() {
		return description;
	}

	public int getCode() {
		return code;
	}

	@Override
	public String toString() {
		return this.getDescription();
	}
}
package voting.client.widgets;

import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;

public class FieldPanel extends Composite {

	private Label fieldLabel;
	private Label valueLabel;

	public FieldPanel(String field, String value) {
		HorizontalPanel panel = new HorizontalPanel();

		initWidget( panel );
		panel.setWidth("100%");

		setStyleName("FieldPanel");
```

```
            fieldLabel = new Label( field );
            fieldLabel.setStyleName("FieldPanel-field");
            fieldLabel.setWordWrap( false );

            valueLabel = new Label( value );
            valueLabel.setStyleName("FieldPanel-value");
            valueLabel.setWordWrap( false );

            HorizontalPanel fieldPanel = new HorizontalPanel();
            fieldPanel.add(fieldLabel);
            fieldPanel.add(valueLabel);
            panel.add(fieldPanel);

            /*valueLabel = new Label( value );
            valueLabel.setStyleName("FieldPanel-value");
            valueLabel.setWordWrap( false );
            HorizontalPanel valuePanel = new HorizontalPanel();
            valuePanel.add(valueLabel);
            valuePanel.setWidth("100%");
            panel.add(valuePanel);*/
        }

        public void setField( String text )
        {
            fieldLabel.setText(text);
        }

        public void setValue( String text )
        {
            valueLabel.setText(text);
        }

}
```

```
package voting.client.widgets;

import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.Hyperlink;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

@SuppressWarnings("deprecation")
public class ListView extends Composite {

	private VerticalPanel mainPanel = new VerticalPanel();

	private TitleCommandBar title;
	private FlexTable rows;

	public ListView( String titleValue, String image, String addLabel ){
		mainPanel.setStyleName("ListView");
		initWidget( mainPanel );
		mainPanel.setWidth("100%");

		title = new TitleCommandBar(titleValue,image);
		title.setStyleName("ListView-titlebar");
		title.setTitleStyle("ListView-titlebar-title");
		mainPanel.add( title );
		getTitleCommandBar().addCommand(addLabel, new ClickListener(){
			public void onClick( Widget sender ) {
				onAdd();
			}
		});
```

```
        createTable();
}
private TitleCommandBar getTitleCommandBar() {
        return title;
}

private void createTable() {
        if (rows != null) {
                mainPanel.remove(rows);
        }
        rows = new FlexTable();
        rows.setWidth("100%");
        mainPanel.add(rows);
        rows.getRowFormatter().setStyleName(0, "ListView-header");
}

protected void clear() {
        createTable();
        onClear();
}

protected void addColumn( String name ){
        int column = rows.getRowCount()>0?rows.getCellCount(0):0;
        Label value = new Label(name);
        value.setStyleName("ListView-header");
        rows.setWidget(0, column, value);
}
protected void addField( int row, String value ){
        row = row + 1;
        int column = 0;
        if( rows.getRowCount()==row ){
                if( row%2 == 0 ) {
```

```
                        rows.getRowFormatter().setStyleName(row,"ListView-
even");
                } else {
                        rows.getRowFormatter().setStyleName(row,"ListView-
odd");
                }
        }
        else{
                column = rows.getCellCount(row);
        }
        Label fieldValue = new Label(value);
        fieldValue.setStyleName("ListView-value");
        rows.setWidget(row, column, fieldValue);
    }
    protected void addRemoveField( int row, String value ){
        final int removeIndex = row;
        row = row + 1;
        int column = 0;
        if( rows.getRowCount()==row ){
                if( row%2 == 0 ) {
                        rows.getRowFormatter().setStyleName(row,"ListView-
even");
                } else {
                        rows.getRowFormatter().setStyleName(row,"ListView-
odd");
                }
        }
        else{
                column = rows.getCellCount(row);
        }

        Hyperlink hyperlink = new Hyperlink( value, "" );
        hyperlink.addClickListener( new ClickListener(){
```

```
                public void onClick( Widget sender ) {
                        onRemove(removeIndex);
                }
        });
        hyperlink.setStyleName("ListView-link");

        rows.setWidget(row, column, hyperlink);
        rows.getColumnFormatter().setWidth(column, "60px");
    }
    protected void addEditField( int row, String value ){
        final int editIndex = row;
        row = row + 1;
        int column = 0;
        if( rows.getRowCount()==row ){
                if( row%2 == 0 ) {
                        rows.getRowFormatter().setStyleName(row,"ListView-
even");
                } else {
                        rows.getRowFormatter().setStyleName(row,"ListView-
odd");
                }
        }
        else{
                column = rows.getCellCount(row);
        }

        Hyperlink hyperlink = new Hyperlink( value, "" );
        hyperlink.addClickListener( new ClickListener(){
                public void onClick( Widget sender ) {
                        onEdit(editIndex);
                }
        });
        hyperlink.setStyleName("ListView-link");
```

```
                rows.setWidget(row, column, hyperlink);
                rows.getColumnFormatter().setWidth(column, "60px");
        }

        protected void onAdd(){}
        protected void onClear(){}
        protected void onRemove(int index){}
        protected void onEdit(int index){}
}
package voting.server;

import java.util.List;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

import voting.model.Voter;

public class MailSender {

        private String fromAddress;

        public MailSender() {
                fromAddress = "vdmuller@gmail.com";
        }
```

```
public boolean sendAddMessage(List<Voter> voters) {
        boolean Result = false;

        if (voters.size() > 0) {
                Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);

        String msgBody = "Você acaba de ser adicionado em uma votação. Assim que a votação se iniciar você receberá um e-mail contendo o link para votar.";

        try {
           Message msg = new MimeMessage(session);
           msg.setFrom(new InternetAddress(fromAddress));

           for (int i=0;i < voters.size();i++) {
                msg.addRecipient(Message.RecipientType.TO,
                      new InternetAddress(voters.get(i).getEmail()));
           }

           msg.setSubject("Você foi adicionado em uma votação");
           msg.setText(msgBody);
           Transport.send(msg);

           Result = true;
        } catch (AddressException e) {
          Result = false;
        } catch (MessagingException e) {
          Result = false;
        }
          }

          return Result;
}
```

```
public boolean sendStartMessage(List<Voter> voters, List<String> auditors) {
        boolean Result = false;

        if (voters.size() > 0) {
                Properties props = new Properties();
            Session session = Session.getDefaultInstance(props, null);

            String msgBody = "Uma votação foi iniciada. Acesse o link para votar: \n";

            try {

              for (int i=0;i < voters.size();i++) {
                    Message msg = new MimeMessage(session);
                     msg.setFrom(new InternetAddress(fromAddress));

                    msg.addRecipient(Message.RecipientType.TO,
                          new InternetAddress(voters.get(i).getEmail()));

                    String                              link                              =
"https://tccvotacao.appspot.com/vote.html?"+voters.get(i).getId();

                    msg.setSubject("Votação iniciada");
                     msg.setText(msgBody+link);
                     Transport.send(msg);
              }

              msgBody = "Uma votação da qual você é auditor acaba de iniciar. Acesse o link para acompanhar: \n";

              for (int i=0;i < auditors.size();i++) {
                    Message msg = new MimeMessage(session);
```

```
                msg.setFrom(new InternetAddress(fromAddress));

                msg.addRecipient(Message.RecipientType.TO,
                        new InternetAddress(auditors.get(i)));

                String link = "http://tccvotacao.appspot.com/auditor.html";

                msg.setSubject("Votação iniciada");
                 msg.setText(msgBody+link);
                 Transport.send(msg);
            }

             Result = true;
        } catch (AddressException e) {
          Result = false;
        } catch (MessagingException e) {
          Result = false;
        }
          }

          return Result;
}

public boolean sendEndMessage(List<Voter> voters, List<String> auditors) {
          boolean Result = false;

          if (voters.size() > 0) {
                  Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);

        String msgBody = "A votação foi encerrada.";

        try {
```

```
                for (int i=0;i < voters.size();i++) {
                        Message msg = new MimeMessage(session);
                         msg.setFrom(new InternetAddress(fromAddress));

                        msg.addRecipient(Message.RecipientType.TO,
                              new InternetAddress(voters.get(i).getEmail()));

                        msg.setSubject("Votação encerrada");
                         msg.setText(msgBody);
                         Transport.send(msg);
                }

                msgBody = "Uma votação da qual você é auditor acaba de encerrar.
Acesse o link para ver o resultado: \n";

                for (int i=0;i < auditors.size();i++) {
                        Message msg = new MimeMessage(session);
                         msg.setFrom(new InternetAddress(fromAddress));

                        msg.addRecipient(Message.RecipientType.TO,
                              new InternetAddress(auditors.get(i)));

                        String link = "http://tccvotacao.appspot.com/auditor.html";

                        msg.setSubject("Votação encerrada");
                         msg.setText(msgBody+link);
                         Transport.send(msg);
                }

                 Result = true;
               } catch (AddressException e) {
                 Result = false;
```

```
                } catch (MessagingException e) {
                  Result = false;
                }
                  }

                  return Result;
        }
}
package voting.client.widgets.dialogs;

import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.HasHorizontalAlignment;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.Widget;

@SuppressWarnings("deprecation")
enum DialogMode {
        ADD, EDIT
}

@SuppressWarnings("deprecation")
public class ObjectDialogBox extends DialogBox {

        private FlexTable fields = new FlexTable();

        public ObjectDialogBox(String string) {
                setHTML( string );
                setWidget( fields );
```

```
        show();
        center();
}
public void addField(String name, String value) {

        TextBox fieldValue = new TextBox();
        fieldValue.setText(value);
        addField( name, fieldValue );
}


public  void addField(String name, Widget fieldValue) {
        int row = fields.getRowCount();
        Label fieldName = new Label( name );
        fieldName.setWordWrap( false );
        fieldName.setStyleName("Dialog-name");
        fieldValue.setStyleName("Dialog-value");
        //fields.getCellFormatter().setWidth(row,1,"100%");
        fields.setWidget(row, 0, fieldName);
        fields.setWidget(row, 1, fieldValue);
}

public void addButtons(){
        int row = fields.getRowCount();
        HorizontalPanel buttons = new HorizontalPanel();
        fields.setWidget(row, 1, buttons );
        buttons.add( new Button( "Ok", new ClickListener(){
                public void onClick( Widget sender ){
                        onSubmit();
                        hide();
                }
        }));
        buttons.add( new Button("Cancel", new ClickListener(){
```

```
                        public void onClick( Widget sender ){
                                hide();
                        }
                });
                fields.getCellFormatter().setHorizontalAlignment(row,                   1,
HasHorizontalAlignment.ALIGN_RIGHT );
        }

        public String getField(int row){
                TextBox field = (TextBox)fields.getWidget(row, 1);
                return field.getText();
        }

        public void onSubmit(){}

}
package voting.model;

import java.io.Serializable;

import javax.jdo.annotations.Extension;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType = IdentityType.APPLICATION,detachable="true")

public class Option implements Serializable {

        /**
         * Serialization ID
```

```
     */
    private static final long serialVersionUID = -3770241673710040 87L;

    @PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
@Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
private String id;

    @Persistent
private String title;

@Persistent
private String description;

@Persistent
private VotingModel votingModel;


public Option() {}

public Option(VotingModel votingModel, String title) {
    this.title = title;
  this.votingModel       = votingModel;
}


public String getId() {
            return id;
    }

    public void setTitle(String title) {
            this.title = title;
    }
```

```
	public String getTitle() {
		return title;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public String getDescription() {
		return description;
	}

	public void setVotingModel(VotingModel votingModel) {
		this.votingModel = votingModel;
	}

	public VotingModel getVotingModel() {
		return votingModel;
	}

}package voting.client.widgets.dialogs;

import voting.client.widgets.OptionListView;
import voting.model.Option;

public class OptionDialog extends ObjectDialogBox {

	private OptionListView view;
	private Option option;
	private DialogMode mode;

	public OptionDialog(OptionListView view) {
```

```
            super("<img src='images/options24.png' hspace='3'>Opcao");
            this.option = new Option();
            this.view = view;
            this.mode = DialogMode.ADD;
            init();
        }

        public OptionDialog(Option option, OptionListView view) {
            super("<img src='images/options24.png' hspace='3'>Opcao");
            this.option = option;
            this.view = view;
            this.mode = DialogMode.EDIT;
            init();
        }

        private void init() {
            addField( "Titulo", option.getTitle() );
            addField( "Descricao", option.getDescription() );
            addButtons();
        }

        public void onSubmit(){
            option.setTitle( getField(0) );
            option.setDescription( getField(1) );
            if (this.mode == DialogMode.ADD) {
                view.add(option);
            }
            else if (this.mode == DialogMode.EDIT) {
                view.refresh();
            }
        }

}
```

```
package voting.client.widgets;

import java.util.List;

import voting.client.widgets.dialogs.OptionDialog;
import voting.model.Option;


public class OptionListView extends ListView {

	private List<Option> list;

	public OptionListView(String titleValue) {
		super(titleValue, "<img src='images/options24.png'>", "Adicionar");
	}

	public void setList(List<Option> list) {
		this.list = list;
		this.refresh();
	}

	public void refresh() {
		clear();

		int row = 0;
		for( java.util.Iterator<Option> it = this.list.iterator(); it.hasNext();){
			Option option = (Option)it.next();
			addField(row,option.getTitle());
			addField(row,option.getDescription());
			addEditField(row,"editar");
			addRemoveField(row,"excluir");
			row++;
		}
```

```
        }

        public void add(Option option) {
                this.list.add(option);
                refresh();
        }

        protected void onClear() {
                addColumn( "Título" );
                addColumn( "Descrição" );
                addColumn( "" );
                addColumn( "" );
        }

        protected void onAdd() {
       new OptionDialog(this);
        }

        protected void onRemove(int index) {
                list.remove(index);
                refresh();
        }

        protected void onEdit(int index) {
                new OptionDialog(list.get(index),this);
        }

}
package voting.model;

import java.io.Serializable;

public class OptionReport implements Serializable {
```

```
	/**
	 *
	 */
	private static final long serialVersionUID = -6639505728975184414L;

	private String id;
    private String title;
    private int votes;


	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getTitle() {
		return title;
	}
	public void setTitle(String title) {
		this.title = title;
	}
	public int getVotes() {
		return votes;
	}
	public void setVotes(int votes) {
		this.votes = votes;
	}

}
package voting.dao;
```

```
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

public final class PMF {
    private static final PersistenceManagerFactory pmfInstance =
        JDOHelper.getPersistenceManagerFactory("transactions-optional");

    private PMF() {}

    public static PersistenceManagerFactory get() {
        return pmfInstance;
    }

    public static PersistenceManager getPM() {
        return pmfInstance.getPersistenceManager();
    }
}package voting.client.widgets;

import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.VerticalPanel;

public class ReportListItem extends Composite {

        private VerticalPanel mainPanel = new VerticalPanel();
        private TitleCommandBar titleBar;
        private FlexTable fields = new FlexTable();

        public ReportListItem(String title) {
                this.mainPanel.setWidth("100%");
                fields.setWidth("100%");
```

```
            titleBar = new TitleCommandBar(title,"");

            mainPanel.add( titleBar );
            mainPanel.add(fields);

            fields.getRowFormatter().setStyleName(0, "AdminListItem-header");
            mainPanel.setStyleName("AdminListItem");

            initWidget( this.mainPanel );
        }

        protected void addField(String name, String value, String style) {
            int row = fields.getRowCount();
            Label fieldName = new Label( name );
            Label fieldValue = new Label( value );
            fieldName.setStyleName("AdminListItem-name");
            fieldValue.setStyleName("AdminListItem-value");
            fields.getCellFormatter().setWidth(row,1,"100%");
            fields.setWidget(row, 0, fieldName);
            fields.setWidget(row, 1, fieldValue);
            if (!style.isEmpty()) {
                fieldName.addStyleName(style);
                fieldValue.addStyleName(style);
            } else {
                int t = row%2;
                fieldName.addStyleName("AdminListItem-
row"+String.valueOf(t));
                fieldValue.addStyleName("AdminListItem-
row"+String.valueOf(t));
            }
        }
```

```
}
package voting.client.widgets;

import voting.client.BaseView;
import voting.model.OptionReport;
import voting.model.Voter;
import voting.model.VoterReport;
import voting.model.VotingReport;
import voting.model.VotingModel.Type;
import voting.util.VUtils;

import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.VerticalPanel;

public class ReportPanel extends Composite {

	private VotingReport report;
	private BaseView view;
	private VerticalPanel mainPanel = new VerticalPanel();

	public ReportPanel(VotingReport report, final BaseView view) {
		this.report = report;
		this.view = view;

		this.mainPanel.setWidth("100%");

		Label lb = new Label("Relatório - "+report.getTitle());
		lb.setStyleName("Report-Title");
		mainPanel.add(lb);
```

```
            buildInfo();
            buildVoters();
            buildResult();

            Button okButton = new Button("Fechar",new ClickHandler() {
                  public void onClick(ClickEvent event) {
                        view.showDefault();
                  }
            });

            mainPanel.add(okButton);

            initWidget( this.mainPanel );
      }

      public void buildInfo() {
            ReportListItem item = new ReportListItem(report.getTitle());

            item.addField("ID", report.getId(), "");
            item.addField("Descrição", report.getDescription(), "");
            item.addField("Tipo", VUtils.TypeToString(report.getType()), "");
            item.addField("Status", VUtils.StatusToString(report.getStatus()), "");
            item.addField("", "", "");
            item.addField("Início", report.getStart().toLocaleString(), "");
            item.addField("Fim", report.getEnd().toLocaleString(), "");

            mainPanel.add(item);
      }

      public void buildVoters() {
            ReportListItem item = new ReportListItem("Quem votou:
"+report.getTotalVoted()+" de "+report.getVoters().size());
```

```
                for (int i=0; i < report.getVoters().size(); i++) {
                        VoterReport aux = report.getVoters().get(i);

                        if (aux.getVoted()) {
                                item.addField(aux.getName(), aux.getEmail(), "");
                        }
                }

                mainPanel.add(item);
        }

        public void buildResult() {
                ReportListItem item = new ReportListItem("Resultado");

                if (report.getType()==Type.OPTION) {
                        for (int i=0; i < report.getOptions().size(); i++) {
                                OptionReport aux = report.getOptions().get(i);

                                item.addField(aux.getTitle(),  aux.getVotes()  +  "  voto(s)",
"");
                        }
                }
                else if (report.getType()==Type.NOMINEE) {
                        for (int i=0; i < report.getVoters().size(); i++) {
                                VoterReport aux = report.getVoters().get(i);

                                if (aux.getVotes() > 0) {
                                        item.addField(aux.getName()     +     "     -     "     +
aux.getEmail(), aux.getVotes() + " voto(s)", "");
                                }
                        }
                }
```

```
                mainPanel.add(item);
        }

}
package voting.client.widgets;

import voting.client.BaseView;
import voting.model.VotingModel;

public class StatusTreeItem extends BaseTreeItem {

        private VotingModel.Status status;

        public StatusTreeItem(String text, VotingModel.Status status, BaseView view)
{
                super(text, view);
                this.status = status;
        }

        public void onTreeItemSelected() {
                getView().showList(this.status);
        }

}
package voting.client.widgets;

import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HasVerticalAlignment;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Hyperlink;
```

```
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

@SuppressWarnings("deprecation")
public class TitleCommandBar extends Composite
{
	private HTML titleLabel;
	private HTML titleImage;
	private HorizontalPanel titlePanel = new HorizontalPanel();
	private Widget lastCommand;
	public TitleCommandBar( String title, String image )
	{
		VerticalPanel panel = new VerticalPanel();

		initWidget( panel );
		panel.setWidth("100%");
		titlePanel.setWidth("100%");
		panel.add(titlePanel);

		titlePanel.setWidth("100%");
		setStyleName("TitleBar");

		titleImage = new HTML( image );
		titlePanel.add( titleImage );
		titlePanel.setCellVerticalAlignment(                    titleImage,
HasVerticalAlignment.ALIGN_MIDDLE );

		titleLabel = new HTML( title );
		titleLabel.setStyleName("TitleBar-title");
		titleLabel.setWordWrap( false );
		titlePanel.add( titleLabel );
		titlePanel.setCellWidth(titleLabel, "100%");
```

```
            titlePanel.setCellVerticalAlignment(                    titleLabel,
HasVerticalAlignment.ALIGN_MIDDLE );
        }
        public void addWidget( Widget widget )
        {
            titlePanel.setCellWidth(titleLabel, "");
            if( lastCommand != null )
                titlePanel.setCellWidth(lastCommand, "");
            lastCommand = widget;
            titlePanel.add( lastCommand );
            titlePanel.setCellWidth(lastCommand, "100%");
            titlePanel.setCellVerticalAlignment(                    lastCommand,
HasVerticalAlignment.ALIGN_MIDDLE );
        }

        public void addCommand( String name, ClickListener command )
        {
            Hyperlink hyperlink = new Hyperlink( name, name );
            hyperlink.addClickListener( command );
            hyperlink.setStyleName("TitleBar-link");
            addWidget( hyperlink );

        }
        public void setText( String text )
        {
            titleLabel.setText(text);
        }
        public void setImage( String image )
        {
            titleImage.setText(image);
        }
        public void setTitleStyle( String style )
        {
```

```
            titleLabel.setStyleName(style);
        }
}
package voting.client.widgets;

import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HasVerticalAlignment;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Hyperlink;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

@SuppressWarnings("deprecation")
public class TitlePanel extends Composite {

        private HTML titleLabel;
        private VerticalPanel mainPanel = new VerticalPanel();
        private HorizontalPanel titlePanel = new HorizontalPanel();

        public TitlePanel(String title) {
            VerticalPanel panel = new VerticalPanel();

            initWidget( panel );
            panel.setWidth("100%");
            mainPanel.setWidth("100%");
            titlePanel.setWidth("100%");
            panel.add(titlePanel);
            panel.add(mainPanel);

            setStyleName("TitlePanel");
```

```
            titleLabel = new HTML( title );
            titleLabel.setStyleName("TitlePanel-title");
            titleLabel.setWordWrap( false );
            titlePanel.add( titleLabel );
        }

        public void addWidget( Widget widget )
        {
            mainPanel.add( widget );
            widget.setWidth("100%");
            mainPanel.setCellWidth(widget, "100%");
            mainPanel.setCellVerticalAlignment(                    widget,
HasVerticalAlignment.ALIGN_MIDDLE );
            widget.setStyleName("TitleBar-content");
        }

        public void addWidget( Widget widget, String style )
        {
            mainPanel.add( widget );
            widget.setWidth("100%");
            mainPanel.setCellWidth(widget, "100%");
            mainPanel.setCellVerticalAlignment(                    widget,
HasVerticalAlignment.ALIGN_MIDDLE );
            widget.setStyleName(style);
        }

        public void addCommand( String name, ClickListener command )
        {
            Hyperlink hyperlink = new Hyperlink( name, name );
            hyperlink.addClickListener( command );
            hyperlink.setStyleName("TitlePanel-link");

            titlePanel.add( hyperlink );
```

```
                hyperlink.setWidth("100%");
                titlePanel.setCellWidth(hyperlink, "100%");
                titlePanel.setCellVerticalAlignment(                    hyperlink,
HasVerticalAlignment.ALIGN_MIDDLE );

        }

        public void setText( String text )
        {
                titleLabel.setText(text);
        }

        public void clear() {
                mainPanel.clear();
        }

}
package voting.client.widgets;

import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HasVerticalAlignment;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Hyperlink;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

public class TopPanel extends Composite {

        private HTML titleLabel;
        private HTML titleImage;
```

```
        private VerticalPanel mainPanel = new VerticalPanel();
        private HorizontalPanel imagePanel = new HorizontalPanel();
        private HorizontalPanel titlePanel = new HorizontalPanel();

        public TopPanel(String title, String image) {
                VerticalPanel p = new VerticalPanel();
                p.setWidth("100%");
                DockPanel dockMainPanel = new DockPanel();
                p.add(dockMainPanel);
                p.setCellWidth(dockMainPanel, "100%");
                initWidget( p );

                VerticalPanel panel = new VerticalPanel();
                dockMainPanel.add(panel,DockPanel.WEST);
                dockMainPanel.setHeight("100%");
                dockMainPanel.setWidth("100%");

                panel.setWidth("100%");
                mainPanel.setWidth("100%");
                titlePanel.setWidth("100%");
                imagePanel.setWidth("100%");
                panel.add(imagePanel);
                panel.add(titlePanel);
                panel.add(mainPanel);

                setStyleName("TitlePanel");
                imagePanel.addStyleName("topPanel-image");

                titleImage = new HTML( image );
                imagePanel.add( titleImage );
                imagePanel.setCellVerticalAlignment(                    titleImage,
HasVerticalAlignment.ALIGN_MIDDLE );
```

```
            titleLabel = new HTML( title );
            titleLabel.setStyleName("TitlePanel-title");
            titleLabel.setWordWrap( false );
            titlePanel.add( titleLabel );
            titlePanel.setCellVerticalAlignment(                    titleLabel,
HasVerticalAlignment.ALIGN_MIDDLE );
        }

        public void addWidget( Widget widget )
        {
            mainPanel.add( widget );
            widget.setWidth("100%");
            mainPanel.setCellWidth(widget, "100%");
            mainPanel.setCellVerticalAlignment(                      widget,
HasVerticalAlignment.ALIGN_MIDDLE );
            widget.setStyleName("TitleBar-content");
        }

        public void addCommand( String name, ClickHandler command )
        {
            Hyperlink hyperlink = new Hyperlink( name, name );
            hyperlink.addClickHandler(command);
            hyperlink.setStyleName("TitlePanel-link");

            titlePanel.add( hyperlink );
            hyperlink.setWidth("100%");
            titlePanel.setCellWidth(hyperlink, "100%");
            titlePanel.setCellVerticalAlignment(                    hyperlink,
HasVerticalAlignment.ALIGN_MIDDLE );
        }

        public void setText( String text )
        {
```

```
            titleLabel.setText(text);
        }

        public void setImage( String image )
        {
            titleImage.setText(image);
        }

        public void clear() {
            mainPanel.clear();
        }

}
package voting.error;

import java.io.Serializable;

public class ValidationException extends Exception implements Serializable {

        /**
         *
         */
        private static final long serialVersionUID = 3473333915184252062L;
        private String message;

        public ValidationException() {
            super();
        }

        public ValidationException(String errorMsg) {
            super();

            this.message = errorMsg;
```

```
	}

	public String getMessage() {
		return "Erro de validação: " + this.message;
	}

}
package voting.client.vote;

import java.util.List;

import voting.client.widgets.FieldPanel;
import voting.client.widgets.TitlePanel;
import voting.error.ErrorMsg;
import voting.error.VotingException;
import voting.model.Option;
import voting.model.Voter;
import voting.model.VotingModel;
import voting.util.VUtils;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.HasHorizontalAlignment;
import com.google.gwt.user.client.ui.HasVerticalAlignment;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RadioButton;
import com.google.gwt.user.client.ui.RootPanel;
```

```
import com.google.gwt.user.client.ui.VerticalPanel;
import
com.google.gwt.user.client.ui.HasVerticalAlignment.VerticalAlignmentConstant;

public class Vote implements EntryPoint {

	private final VoteServiceAsync voteService = (VoteServiceAsync) GWT
	.create(VoteService.class);

	private VerticalPanel mainPanel = new VerticalPanel();

	private String id;
	private Voter voter;
	private VotingModel votingModel;

	private String optionId;

	public void onModuleLoad() {
		VerticalPanel panel = new VerticalPanel();

	panel.setHorizontalAlignment(HasHorizontalAlignment.ALIGN_CENTER);
		panel.setHeight("100%");
		panel.setWidth("100%");
		panel.add(mainPanel);

		mainPanel.addStyleName("mainPanel");
	   mainPanel.setWidth("600px");

		RootPanel.get("maincontainer").add(panel);

		this.optionId = "";

		try {
```

```
                this.id =  Window.Location.getQueryString().substring(1);
                //this.id                                                    =
"agp0Y2N2b3RhY2FvchwLEgtWb3RpbmdNb2RlbBgSDAsSBVZvdGVyGBQM";
                loadVoter();
            } catch (Exception e) {

            }
        }

        public void loadVoter() {
            voteService.getVoter(id,
                    new AsyncCallback<Voter>() {
                        public void onFailure(Throwable caught) {
                            showError(caught.getMessage());
                        }

                        public void onSuccess(Voter result) {
                            voter = result;
                            votingModel = voter.getVotingModel();
                            show();
                        }
                    });
        }

        public void setOption(String id) {
            this.optionId = id;
        }

        public String getOption() {
            return this.optionId;
        }

        public void showError(String error) {
```

```
        mainPanel.clear();
        TitlePanel tpErro = new TitlePanel("Erro!");
        mainPanel.add(tpErro);

        Label lbDesc = new Label(error);
        tpErro.addWidget(lbDesc);
}

protected void onVote() {
        voteService.vote(this.voter.getId(),this.optionId,
                        new AsyncCallback<Voter>() {
                                public void onFailure(Throwable caught) {
                                        showError(caught.getMessage());
                                }

                                public void onSuccess(Voter result) {
                                        voter = result;
                                        votingModel = voter.getVotingModel();
                                        show();
                                }
                        });
}

public void show() {
        mainPanel.clear();
        showVoting();
        showVoter();
        showOptions();
}

@SuppressWarnings("deprecation")
protected void showVoting() {
        TitlePanel tpVoting = new TitlePanel(votingModel.getTitle());
```

```
            mainPanel.add(tpVoting);

            Label lbDesc = new Label(votingModel.getDescription());
            tpVoting.addWidget(lbDesc,"VoteLabel");

            FieldPanel fpType = new
FieldPanel("Tipo",VUtils.TypeToString(votingModel.getType()));
            tpVoting.addWidget(fpType);

            FieldPanel fpStart = new
FieldPanel("Início",votingModel.getStart().toLocaleString());
            tpVoting.addWidget(fpStart);

            FieldPanel fpEnd = new
FieldPanel("Fim",votingModel.getEnd().toLocaleString());
            tpVoting.addWidget(fpEnd);

            FieldPanel fpStatus = new
FieldPanel("Status",votingModel.getStatus().toString());
            tpVoting.addWidget(fpStatus);
        }

        protected void showVoter() {
            TitlePanel tpVoter = new TitlePanel(voter.getName());
            mainPanel.add(tpVoter);

            Label lbEmail= new Label(voter.getEmail());
            tpVoter.addWidget(lbEmail,"VoteLabel");

            FieldPanel fpVoted = new
FieldPanel("Votou",VUtils.BooleanToString(voter.voted()));
            tpVoter.addWidget(fpVoted);
```

```
            FieldPanel fpVotable = new
FieldPanel("Elegível",VUtils.BooleanToString(voter.isVotable()));
            tpVoter.addWidget(fpVotable);
        }

        protected void showOptions() {
            if (!voter.voted()) {
                TitlePanel tpOptions = new TitlePanel("Opções");
                mainPanel.add(tpOptions);

                if (votingModel.getType() == VotingModel.Type.OPTION) {
                    List<Option> options = votingModel.getOptions();
                    for (int i = 0; i < options.size(); i++) {
                        HorizontalPanel hp = new HorizontalPanel();
                        TitlePanel tpOption = new
TitlePanel(options.get(i).getTitle());
                        Label lbDesc = new
Label(options.get(i).getDescription());
                        tpOption.addWidget(lbDesc,"VoteLabel");
                        RadioButton rbDesc = new
RadioButton("option","");

                        final String id = options.get(i).getId();
                        rbDesc.addClickHandler(new ClickHandler() {
                            @Override
                            public void onClick(ClickEvent event) {
                                setOption(id);
                            }
                        });

                        hp.add(rbDesc);
                        hp.add(tpOption);
                        hp.setCellWidth(tpOption, "100%");
```

```
                        hp.setCellVerticalAlignment(rbDesc,
HasVerticalAlignment.ALIGN_MIDDLE);
                        tpOptions.addWidget(hp);
                    }
                } else {
                    List<Voter> voters = votingModel.getVoters();
                    for (int i = 0; i < voters.size(); i++) {
                        if (voters.get(i).isVotable()) {
                            TitlePanel tpOption = new
TitlePanel(voters.get(i).getName());
                            RadioButton rbDesc = new
RadioButton("option",voters.get(i).getEmail());

                            final String id = voters.get(i).getId();
                            rbDesc.addClickHandler(new ClickHandler()
{
                                @Override
                                public void onClick(ClickEvent event) {
                                    setOption(id);
                                }
                            });

                            tpOption.addWidget(rbDesc);
                            tpOptions.addWidget(tpOption);
                        }
                    }
                }

                HorizontalPanel panel = new HorizontalPanel();
            Button okButton = new Button("Votar",new ClickHandler() {
                    public void onClick(ClickEvent event) {
                        onVote();
                    }
```

```
                });

                panel.setStyleName("AdminVotingPanel-buttons");
                panel.add(okButton);

panel.setCellHorizontalAlignment(okButton,HasHorizontalAlignment.ALIGN_CENTE
R);

                tpOptions.addWidget(panel);
            }
        }



}
package voting.client.widgets.dialogs;

import voting.client.widgets.VoterListView;
import voting.model.Voter;

public class VoterDialog extends ObjectDialogBox {

        private VoterListView view;
        private Voter voter;
        private DialogMode mode;

        public VoterDialog(VoterListView view) {
                super("<img src='images/voters24.png' hspace='3'>Eleitor");
                this.voter = new Voter();
                this.view = view;
                this.mode = DialogMode.ADD;
                init();
        }
```

```
	private void init() {
		addField( "Nome", voter.getName() );
		addField( "E-mail", voter.getEmail() );
		addButtons();
	}

	public void onSubmit(){
		voter.setName( getField(0) );
		voter.setEmail( getField(1) );
		if (this.mode == DialogMode.ADD) {
			view.add(voter);
		}
	}

}
package voting.client.widgets;

import java.util.List;

import voting.client.widgets.dialogs.VoterDialog;
import voting.model.Voter;

public class VoterListView extends ListView {

	private List<Voter> list;

	public VoterListView(String titleValue) {
		super(titleValue, "<img src='images/voters24.png'>", "Adicionar");
	}

	public void setList(List<Voter> list) {
		this.list = list;
```

```
        this.refresh();
}

public void refresh() {
        clear();

        int row = 0;
        for( java.util.Iterator<Voter> it = this.list.iterator(); it.hasNext();){
                Voter voter = (Voter)it.next();
                addField(row,voter.getName());
                addField(row,voter.getEmail());
                addRemoveField(row,"excluir");
                row++;
        }
}

public void add(Voter voter) {
        this.list.add(voter);
        refresh();
}

protected void onClear() {
        addColumn( "Nome" );
        addColumn( "E-mail" );
        addColumn( "" );
}

protected void onAdd() {
        new VoterDialog(this);
}

protected void onRemove(int index) {
        list.remove(index);
```

```
                refresh();
        }

}
package voting.client.vote;

import voting.error.VotingException;
import voting.model.Voter;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("vote")
public interface VoteService extends RemoteService {

        Voter getVoter(String key) throws VotingException;
        Voter vote(String voterId,String optionId) throws VotingException;

}
package voting.client.vote;

import voting.model.Voter;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface VoteServiceAsync {

        void getVoter(String key, AsyncCallback<Voter> callback);

        void vote(String voterId, String optionId, AsyncCallback<Voter> callback);

}
package voting.server.vote;
```

```
import javax.servlet.ServletException;

import voting.client.vote.VoteService;
import voting.dao.PMF;
import voting.dao.VotingDAO;
import voting.error.VotingException;
import voting.error.ErrorMsg;
import voting.model.VotingModel;
import voting.model.Vote;
import voting.model.Voter;

import com.google.appengine.repackaged.org.apache.commons.logging.Log;
import com.google.appengine.repackaged.org.apache.commons.logging.LogFactory;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class VoteServiceImpl extends RemoteServiceServlet implements VoteService
{

	/**
	 *
	 */
	private static final long serialVersionUID = -3655245234592586705L;

	/**
	 * Log channel
	 */
	private static Log _log = LogFactory.getLog(VoteServiceImpl.class);


	@Override
	public void init() throws ServletException
	{
```

```
        super.init();
    }

    public Voter getVoter(String key) throws VotingException {
        Voter result = null;
        VotingDAO dao = new VotingDAO(PMF.get());

        try {
            result = dao.loadVoter(key);
        }
        catch (Exception e) {
            _log.error(e.getMessage(), e);
            throw new VotingException(ErrorMsg.DATABASE);
    }

        if (result.getVotingModel().getStatus() !=
VotingModel.Status.RUNNING) {
            ErrorMsg error = ErrorMsg.DEFAULT;
            switch (result.getVotingModel().getStatus()) {
                case FINISHED:
                    error = ErrorMsg.VOTING_FINISHED;
                    break;
                case WAITING:
                    error = ErrorMsg.VOTING_WAITING;
                    break;
            }
            throw new VotingException(error);
        }

        return result;
    }

    @Override
```

```
    public Voter vote(String voterId, String optionId) throws VotingException {
        //TODO: Verificar se ja votou

        Voter voter = null;
        VotingDAO dao = new VotingDAO(PMF.get());

        try {
            voter = dao.loadVoter(voterId);
        }
        catch (Exception e) {
            _log.error(e.getMessage(), e);
            throw new VotingException(ErrorMsg.DATABASE);
    }

        if (voter.getVotingModel().getStatus() != VotingModel.Status.RUNNING)
{
            ErrorMsg error = ErrorMsg.DEFAULT;
            switch (voter.getVotingModel().getStatus()) {
                case FINISHED:
                    error = ErrorMsg.VOTING_FINISHED;
                    break;
                case WAITING:
                    error = ErrorMsg.VOTING_WAITING;
                    break;
            }
            throw new VotingException(error);
        }

        Vote vote = new Vote();
        voter.setVote(vote);

        if (voter.getVotingModel().getType() == VotingModel.Type.OPTION) {
            vote.setOption(optionId);
```

```
                }
                else if (voter.getVotingModel().getType() ==
VotingModel.Type.NOMINEE) {
                        vote.setNominee(optionId);
                }

                try {
                        voter = dao.storeVoter(voter);
                }
                catch (Exception e) {
                        _log.error(e.getMessage(), e);
                        throw new VotingException(ErrorMsg.DATABASE);
        }

                return voter;
          }

}
package voting.server;

import java.io.IOException;
import java.util.Date;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import voting.dao.PMF;
import voting.dao.VotingDAO;
import voting.model.VotingModel;
import voting.server.admin.AdminServiceImpl;
```

```
import com.google.appengine.repackaged.org.apache.commons.logging.Log;
import com.google.appengine.repackaged.org.apache.commons.logging.LogFactory;

@SuppressWarnings("serial")
public class VotingCron extends HttpServlet {

	/**
	 * Log channel
	 */
	private static Log _log = LogFactory.getLog(AdminServiceImpl.class);

	private VotingDAO dao = new VotingDAO(PMF.get());
	private Date now;

	protected void service (HttpServletRequest req, HttpServletResponse res)
					throws ServletException, IOException
{
			now = new Date();
			startVotings();
			endVotings();
	}

	protected void startVotings() {
		try {
			List<VotingModel> list =
dao.getVotings(VotingModel.Status.WAITING);

			for (int i=0;i < list.size();i++) {
			VotingModel v = list.get(i);

			if (checkDate(v.getStart())) {
				v.setStatus(VotingModel.Status.RUNNING);
```

```
                    v = dao.storeVoting(v);

                    MailSender mail = new MailSender();
                mail.sendStartMessage(v.getVoters(),v.getAuditors());
                }
            }

            } catch (Exception e) {
                _log.error(e.getMessage(), e);
            }
        }

        protected void endVotings() {
            try {
                List<VotingModel> list =
dao.getVotings(VotingModel.Status.RUNNING);

                for (int i=0;i < list.size();i++) {
                VotingModel v = list.get(i);

                if (checkDate(v.getEnd())) {
                    v.setStatus(VotingModel.Status.FINISHED);
                    v = dao.storeVoting(v);

                    MailSender mail = new MailSender();
                mail.sendEndMessage(v.getVoters(),v.getAuditors());
                }
            }

            } catch (Exception e) {
                _log.error(e.getMessage(), e);
            }
        }
```

```
	@SuppressWarnings("deprecation")
	protected boolean checkDate(Date d) {
		boolean result = false;

		if (d.getYear() == this.now.getYear()) {
			if (d.getMonth() == this.now.getMonth()) {
				if (d.getDate() == this.now.getDate()) {
					if (d.getHours() == this.now.getHours()) {
						if (d.getMinutes() == this.now.getMinutes()) {
							result = true;
						}
					}
				}
			}
		}

		return result;
	}
}
package voting.dao;

import java.util.List;

import javax.jdo.FetchPlan;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;
import javax.jdo.Query;
import javax.jdo.Transaction;

import voting.model.Option;
import voting.model.OptionReport;
import voting.model.Voter;
```

```
import voting.model.VoterReport;
import voting.model.VotingModel;
import voting.model.VotingReport;
import voting.model.VotingModel.Status;
import voting.model.VotingModel.Type;

public class VotingDAO {

	private PersistenceManagerFactory pmf;

	public VotingDAO(PersistenceManagerFactory pmf) {
		this.pmf = pmf;
	}

	/** Accessor for a PersistenceManager */
   protected PersistenceManager getPersistenceManager()
   {
      return pmf.getPersistenceManager();
   }

	public VotingModel storeVoting(VotingModel v) throws Exception {
		PersistenceManager pm = getPersistenceManager();
		pm.setDetachAllOnCommit(true);
		pm.getFetchPlan().addGroup(FetchPlan.ALL);
		Transaction tx = pm.currentTransaction();

		try {
			tx.begin();

			v = pm.makePersistent(v);

			tx.commit();
		}
```

```
            catch (Exception e) {
                    if (tx.isActive()) {
        tx.rollback();
       }
                    throw e;
    }
            finally {
      pm.close();
    }

            return v;
      }

      public void deleteVoting(String id) throws Exception {
            PersistenceManager pm = getPersistenceManager();
            Transaction tx = pm.currentTransaction();

            try {
                    tx.begin();

                    VotingModel votingModel =
pm.getObjectById(VotingModel.class,id);
                    pm.deletePersistent(votingModel);

                    tx.commit();
            }
            catch (Exception e) {
                    if (tx.isActive()) {
        tx.rollback();
       }
                    throw e;
    }
            finally {
```

```
  pm.close();
}
  }

  public void deleteVoting(VotingModel v) throws Exception {
        PersistenceManager pm = getPersistenceManager();
        Transaction tx = pm.currentTransaction();

        try {
              tx.begin();

              pm.deletePersistent(v);

              tx.commit();
        }
        catch (Exception e) {
              if (tx.isActive()) {
     tx.rollback();
   }
              throw e;
}
        finally {
  pm.close();
}
  }

  public VotingModel loadVoting(String id) throws Exception {
        PersistenceManager pm = getPersistenceManager();
        pm.setDetachAllOnCommit(true);
        pm.getFetchPlan().addGroup(FetchPlan.ALL);
        Transaction tx = pm.currentTransaction();

        VotingModel result = null;
```

```
            try {
                    tx.begin();

                    result = pm.getObjectById(VotingModel.class,id);

                    tx.commit();
            }
            catch (Exception e) {
                    if (tx.isActive()) {
        tx.rollback();
     }
                    throw e;
    }
            finally {
      pm.close();
    }

            return result;
      }

      @SuppressWarnings("unchecked")
      public List<VotingModel> getVotings(VotingModel.Status filter) throws
Exception {
            List<VotingModel> result;
            PersistenceManager pm = getPersistenceManager();
            pm.setDetachAllOnCommit(true);
            Transaction tx = pm.currentTransaction();

            try {
                    if (filter == VotingModel.Status.ALL) {
                            tx.begin();
                            Query query = pm.newQuery(VotingModel.class);
```

```
                        result = (List<VotingModel>) query.execute();
                        tx.commit();
                    } else {
                        tx.begin();
                        Query query = pm.newQuery(VotingModel.class,"status
== statusParam");
                        //query.declareImports("import
voting.model.VotingModel");
                        query.declareParameters("int statusParam");
                        result = (List<VotingModel>) query.execute(filter);
                        tx.commit();
                    }
                }
                catch (Exception e) {
                    if (tx.isActive()) {
            tx.rollback();
          }
                    throw e;
        }
                finally {
          pm.close();
        }

            return result;
          }

          @SuppressWarnings("unchecked")
          public List<VotingModel> getAuditorVotings(String email, VotingModel.Status
filter) throws Exception {
                List<VotingModel> result;
                PersistenceManager pm = getPersistenceManager();
                pm.setDetachAllOnCommit(true);
                Transaction tx = pm.currentTransaction();
```

```
                try {
                        if (filter == VotingModel.Status.ALL) {
                                tx.begin();
                                Query query = pm.newQuery(VotingModel.class,"auditors
== auditorParam");
                                query.declareParameters("String auditorParam");
                                result = (List<VotingModel>) query.execute(email);
                                tx.commit();
                        } else {
                                tx.begin();
                                Query query = pm.newQuery(VotingModel.class);
                                query.setFilter("status == statusParam && auditors ==
auditorParam");
                                query.declareParameters("int statusParam, String
auditorParam");
                                result = (List<VotingModel>) query.execute(filter,email);
                                tx.commit();
                        }
                }
                catch (Exception e) {
                        if (tx.isActive()) {
            tx.rollback();
          }
                        throw e;
      }
                finally {
          pm.close();
      }

            return result;
        }
```

```
  public Voter loadVoter(String id) throws Exception {
        PersistenceManager pm = getPersistenceManager();
        pm.setDetachAllOnCommit(true);
        pm.getFetchPlan().addGroup(FetchPlan.ALL);
        pm.getFetchPlan().setMaxFetchDepth(-1);
        Transaction tx = pm.currentTransaction();

        Voter result = null;

        try {
              tx.begin();

              result = pm.getObjectById(Voter.class,id);

              tx.commit();
        }
        catch (Exception e) {
              if (tx.isActive()) {
     tx.rollback();
   }
              throw e;
}
        finally {
  pm.close();
}

        return result;
  }

  public Option loadOption(String id) throws Exception {
        PersistenceManager pm = getPersistenceManager();
        pm.setDetachAllOnCommit(true);
        pm.getFetchPlan().addGroup(FetchPlan.ALL);
```

```
            Transaction tx = pm.currentTransaction();

            Option result = null;

            try {
                    tx.begin();

                    result = pm.getObjectById(Option.class,id);

                    tx.commit();
            }
            catch (Exception e) {
                    if (tx.isActive()) {
        tx.rollback();
    }
                    throw e;
}
            finally {
  pm.close();
}

            return result;
  }

  public Voter storeVoter(Voter voter) throws Exception {
            PersistenceManager pm = getPersistenceManager();
            pm.setDetachAllOnCommit(true);
            Transaction tx = pm.currentTransaction();

            try {
                    tx.begin();

                    voter = pm.makePersistent(voter);
```

```
                tx.commit();
            }
            catch (Exception e) {
                if (tx.isActive()) {
        tx.rollback();
    }
                throw e;
}
            finally {
  pm.close();
}

            return voter;
  }

  public VotingReport getVotingReport(String id) throws Exception {
            VotingReport result = new VotingReport();

            PersistenceManager pm = getPersistenceManager();
            Transaction tx = pm.currentTransaction();

            VotingModel v = null;

            try {
                tx.begin();

                v = pm.getObjectById(VotingModel.class,id);

                //Monta o objeto de relatorio
                result.setTitle(v.getTitle());
                result.setDescription(v.getDescription());
                result.setEnd(v.getEnd());
```

```
result.setId(v.getId());
result.setStart(v.getStart());
result.setStatus(v.getStatus());
result.setType(v.getType());
result.setTotalVoted(0);

for (int i=0; i < v.getOptions().size(); i++) {
        Option aux = v.getOptions().get(i);
        OptionReport or = new OptionReport();

        or.setId(aux.getId());
        or.setTitle(aux.getTitle());
        or.setVotes(0);

        result.getOptions().add(or);
}

for (int i=0; i < v.getVoters().size(); i++) {
        Voter aux = v.getVoters().get(i);
        VoterReport vr = new VoterReport();

        vr.setEmail(aux.getEmail());
        vr.setId(aux.getId());
        vr.setName(aux.getName());
        vr.setVoted(aux.getVote()!=null);
        vr.setVotes(0);

        if (vr.getVoted()) {
                result.setTotalVoted(result.getTotalVoted()+1);
        }

        if (v.getStatus()==Status.FINISHED) {
                if (v.getType()==Type.OPTION) {
```

```
                                        if (vr.getVoted()) {
                                            for (int j=0; j < result.getOptions().size(); j++) {
                                                OptionReport aux2 = result.getOptions().get(i);
                                                if (aux2.getId().equals(aux.getVote().getOption())) {

        aux2.setVotes(aux2.getVotes() + 1);
                                                    break;
                                                }
                                            }
                                        }
                                    }
                                    else if (v.getType()==Type.NOMINEE) {
                                        for (int j=0; j < v.getVoters().size(); j++) {
                                            Voter aux2 = v.getVoters().get(i);
                                            if ((aux.getVote() != null) && (aux2.getId().equals(aux.getVote().getNominee()))) {
                                                vr.setVotes(vr.getVotes() + 1);
                                            }
                                        }
                                    }
                                }

                                result.getVoters().add(vr);
                            }

                            tx.commit();
                        }
                        catch (Exception e) {
                            if (tx.isActive()) {
                    tx.rollback();
```

```
            }
                        throw e;
        }
                finally {
            pm.close();
        }

                return result;
            }

}
package voting.error;

import java.io.Serializable;

public class VotingException extends Exception implements Serializable {

        /**
         *
         */
        private static final long serialVersionUID = -8497339472520350711L;
        private ErrorMsg error;

        public VotingException() {
                super();
        }

        public VotingException(ErrorMsg errorMsg) {
                super();

                this.error = errorMsg;
        }
```

```
        public ErrorMsg getError2() {
                return this.error;
        }

        public String getMessage() {
                return this.error.toString();
        }

}
package voting.util;

import voting.model.VotingModel;

public final class VUtils {

        public static String fcase( String palavra ) {
      if( palavra != null ) {
         int len = palavra.length();
         String out = "";
         out += palavra.substring( 0, 1 ).toUpperCase();
         out += palavra.substring( 1,len ).toLowerCase();
         return out;
      }
      return palavra;
   }

        public static String TypeToString(VotingModel.Type type) {
                String result = "";

                switch (type) {
                case OPTION:
                        result = "Enquete";
                        break;
```

```
            case NOMINEE:
                result = "Eleicao";
                break;
            default:
                result = "";
                break;
            }

            return result;
    }

    public static String StatusToString(VotingModel.Status status) {
            String result = "";

            switch (status) {
            case FINISHED:
                result = "FINALIZADA";
                break;
            case RUNNING:
                result = "EM ANDAMENTO";
                break;
            case WAITING:
                result = "NÃO INICIADA";
                break;
            default:
                result = "";
                break;
            }

            return result;
    }

    public static String BooleanToString(boolean bool) {
```

```
                String result = "Nao";

                if (bool) {
                        result = "Sim";
                }

                return result;
        }

}
package voting.model;

import java.io.Serializable;

import javax.jdo.annotations.Extension;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType = IdentityType.APPLICATION,detachable="true")

public class Option implements Serializable {

        /**
         * Serialization ID
         */
        private static final long serialVersionUID = -3770241673710040087L;

        @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    @Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
```

```
private String id;

    @Persistent
private String title;

@Persistent
private String description;

@Persistent
private VotingModel votingModel;


public Option() {}

public Option(VotingModel votingModel, String title) {
    this.title = title;
  this.votingModel      = votingModel;
}


public String getId() {
            return id;
    }

    public void setTitle(String title) {
            this.title = title;
    }

    public String getTitle() {
            return title;
    }

    public void setDescription(String description) {
```

```
		this.description = description;
	}

	public String getDescription() {
		return description;
	}

	public void setVotingModel(VotingModel votingModel) {
		this.votingModel = votingModel;
	}

	public VotingModel getVotingModel() {
		return votingModel;
	}

}package voting.model;

import java.io.Serializable;

public class OptionReport implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = -6639505728975184414L;

	private String id;
   private String title;
   private int votes;


	public String getId() {
		return id;
```

```
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getTitle() {
		return title;
	}
	public void setTitle(String title) {
		this.title = title;
	}
	public int getVotes() {
		return votes;
	}
	public void setVotes(int votes) {
		this.votes = votes;
	}

}
package voting.model;

import java.io.Serializable;

import javax.jdo.annotations.Extension;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType = IdentityType.APPLICATION,detachable="true")

public class Vote implements Serializable {
```

```
	/**
	 * Serialization ID
	 */
	private static final long serialVersionUID = 25029900782727670 04L;

	@PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
@Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
private String id;

	@Persistent(mappedBy="vote")
private Voter voter;

private String option;

private String nominee;


public Vote() {}

public String getId() {
		return id;
	}

	public void setVoter(Voter voter) {
		this.voter = voter;
	}

	public Voter getVoter() {
		return voter;
	}

	public void setOption(String option) {
```

```
		this.option = option;
	}

	public String getOption() {
		return option;
	}

	public void setNominee(String nominee) {
		this.nominee = nominee;
	}

	public String getNominee() {
		return nominee;
	}

}package voting.model;

import java.io.Serializable;

import javax.jdo.annotations.Extension;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType = IdentityType.APPLICATION,detachable="true")

public class Voter implements Serializable {

	/**
	 * Serialization ID
	 */
```

```
    private static final long serialVersionUID = -3770241673710040087L;

    @PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
@Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
private String id;

    @Persistent
private String name;

@Persistent
private String email;

@Persistent
private boolean votable;

@Persistent
private VotingModel votingModel;

@Persistent
private Vote vote;


public Voter() {}

public Voter(VotingModel votingModel, String name, String email) {
    this.name = name;
  this.email = email;
  this.votable    = true;
  this.votingModel       = votingModel;
}
```

```
public String getId() {
		return id;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public boolean isVotable() {
		return votable;
	}

	public void setVotable(boolean votable) {
		this.votable = votable;
	}

	public void setVotingModel(VotingModel votingModel) {
		this.votingModel = votingModel;
	}

	public VotingModel getVotingModel() {
		return votingModel;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getEmail() {
```

```
		return email;
	}

	public boolean voted() {
		return this.vote != null;
	}

	public void setVote(Vote vote) {
		if (vote != null) {
			vote.setVoter(this);
			this.vote = vote;
		}
	}

	public Vote getVote() {
		return vote;
	}

}package voting.model;

import java.io.Serializable;

public class VoterReport implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 8321277357249315056L;

	private String id;
   private String name;
   private String email;
   private boolean voted;
```

```
private int votes;


public boolean getVoted() {
        return voted;
    }

    public void setVoted(boolean voted) {
        this.voted = voted;
    }

    public int getVotes() {
        return votes;
    }

    public void setVotes(int votes) {
        this.votes = votes;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
```

```
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getEmail() {
		return email;
	}

}
package voting.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.jdo.annotations.Extension;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType = IdentityType.APPLICATION,detachable="true")

public class VotingModel implements Serializable {

	public enum Type { OPTION, NOMINEE }
	public enum Status { ALL, WAITING, RUNNING, FINISHED }

	/**
```

```
     * Serialization ID
     */
    private static final long serialVersionUID = 4867045171996614194L;

    /*
     * O GILEAD precisa que a PrimaryKey receba o nome "id" para funcionar
     */
    @PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
@Extension(vendorName="datanucleus", key="gae.encoded-pk", value="true")
private String id;

@Persistent
private String title;

@Persistent
private String description;

@Persistent
private Type type;

@Persistent
private Status status;

@Persistent
@Extension(vendorName = "datanucleus", key = "is-second-class", value="false")
private Date date;

@Persistent
@Extension(vendorName = "datanucleus", key = "is-second-class", value="false")
private Date start;

@Persistent
```

```
@Extension(vendorName = "datanucleus", key = "is-second-class", value="false")
private Date end;

@Persistent(mappedBy = "votingModel")
private List<Voter> voters;

@Persistent(mappedBy = "votingModel")
private List<Option> options;

@Persistent
private ArrayList<String> auditors;


public VotingModel() {
      this.date               = new Date();
      this.start              = new Date();
      this.end                = new Date();
      this.voters     = new ArrayList<Voter>();
      this.options    = new ArrayList<Option>();
      this.auditors   = new ArrayList<String>();
      this.status     = Status.WAITING;
      this.type = Type.OPTION;
}

public VotingModel(String title, Type type) {
   this.type               = type;
   this.title          = title;
   this.date               = new Date();
   this.start              = new Date();
     this.end              = new Date();
   this.status       = Status.WAITING;
   this.type = Type.OPTION;
```

```
    this.voters       = new ArrayList<Voter>();
    this.options      = new ArrayList<Option>();
    this.auditors     = new ArrayList<String>();
}


public String getId() {
        return id;
    }

    public void setType(Type type) {
        this.type = type;
    }

    public Type getType() {
        return type;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getDescription() {
        return description;
    }
```

```
public void setDate(Date date) {
        this.date = date;
}

public Date getDate() {
        return date;
}

public void setStart(Date start) {
        this.start = start;
}

public Date getStart() {
        return start;
}

public void setEnd(Date end) {
        this.end = end;
}

public Date getEnd() {
        return end;
}

public void setStatus(Status status) {
        this.status = status;
}

public Status getStatus() {
        return status;
}
```

```
    public void setVoters(List<Voter> voter) {
            this.voters = voter;
    }

    public List<Voter> getVoters() {
            return voters;
    }

    public void addVoter(Voter p) {
   p.setVotingModel(this);
   this.voters.add(p);
}

    public void removeVoter(Voter p) {
   this.voters.remove(p);
}

    public void setOptions(List<Option> options) {
            this.options = options;
    }

    public List<Option> getOptions() {
            return options;
    }

    public void setAuditors(ArrayList<String> auditors) {
            this.auditors = auditors;
    }

    public ArrayList<String> getAuditors() {
            return auditors;
    }
```

```
        public void addAuditor(String email) {
       this.auditors.add(email);
   }

}package voting.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import voting.model.VotingModel.Status;
import voting.model.VotingModel.Type;

public class VotingReport implements Serializable {

        /**
         *
         */
        private static final long serialVersionUID = 7899649212261686976L;

        private String id;
        private String title;
        private String description;
        private Type type;
        private Status status;
        private Date start;
        private Date end;
        private int totalVoted;
        private List<VoterReport> voters;
        private List<OptionReport> options;
```

```
    public VotingReport() {
    this.voters      = new ArrayList<VoterReport>();
    this.options     = new ArrayList<OptionReport>();
}

    public void setId(String id) {
            this.id = id;
    }

    public String getId() {
            return id;
    }

    public void setTitle(String title) {
            this.title = title;
    }

    public String getTitle() {
            return title;
    }

    public void setDescription(String description) {
            this.description = description;
    }

    public String getDescription() {
            return description;
    }

    public void setType(Type type) {
            this.type = type;
    }
```

```
public Type getType() {
	return type;
}

public void setStatus(Status status) {
	this.status = status;
}

public Status getStatus() {
	return status;
}

public void setStart(Date start) {
	this.start = start;
}

public Date getStart() {
	return start;
}

public void setEnd(Date end) {
	this.end = end;
}

public Date getEnd() {
	return end;
}

public void setVoters(List<VoterReport> voters) {
	this.voters = voters;
}

public List<VoterReport> getVoters() {
```

```
        return voters;
    }

    public void setOptions(List<OptionReport> options) {
        this.options = options;
    }

    public List<OptionReport> getOptions() {
        return options;
    }

    public void setTotalVoted(int totalVoted) {
        this.totalVoted = totalVoted;
    }

    public int getTotalVoted() {
        return totalVoted;
    }

}
```