UNIVERSIDADE FEDERAL DE SANTA CATARINA

Desenvolvimento de uma aplicação para monitoração de informações em tempo real, associadas ao transporte público

LUCAS THUM SILVEIRA SCHMIDT

FLORIANÓPOLIS, 2012

Lucas Thum Silveira Schmidt

Desenvolvimento de uma aplicação para monitoração de informações em tempo real, associadas ao transporte público

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

Orientador: Prof. Dr. Fernando Augusto da Silva Cruz

Florianópolis,

2012

RESUMO

Esse trabalho descreve o desenvolvimento de uma aplicação web para assistir a monitoração de dados em tempo real associados ao transporte público. Informações como a localização de um ônibus, mensagens a respeito do estado do serviço de transporte público e alterações na agenda desse podem ser armazenadas e disponibilizadas para o público. Através da obtenção e distribuição de dados por meio da especificação GTFS-realtime, os mesmos podem ser transmitidos para serviços como o Google Transit, que terá a função de apresentar esses aos seus usuários. As informações em tempo real podem ser obtidas diretamente de veículos associados a agências de transporte público, transmitidas para um servidor e disponibilizadas por meio de serviços-web. Com o planejamento e o desenvolvimento de uma aplicação servidor e uma aplicação para ser executada em um veículo, conseguiu-se mostrar o funcionamento desse ecossistema.

Palavras-Chave: Transporte Público. GTFS. GTFS-realtime. Google Transit. Tempo Real. Serviços-web. Dispositivos Móveis.

ABSTRACT

This work describes the development of a web application to help the monitoring of real-time data related to public transportation. Information like the location of a bus, messages regarding the state of a service and changes in the schedule of public transportation can be stored and made available to the public. If data is obtained and distributed thought GTFS-real-time specification, these data can be broadcasted to services like Google Transit, that should present this data to its users. The real-time information can be obtained directly from vehicles associated to public transportation agencies, broadcasted to a server and made available with web-services. With the planning and the development of a server application and another one to be executed in a vehicle, it was possible to demonstrate the ecosystem working.

Keywords: Public Transportation. GTFS. GTFS-realtime. Google Transit. Real-time. Web-services. Mobile devices.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

GTFS – General Transit Feed Specification (Especificação de dados de alimentação de transporte público da Google)
JSON - JavaScript Object Notation
MVC – Model View Controller
MVP – Model View Presenter
GWT - Google Web Toolkit
REST – Representational State Transfer (Transferência de Estado Representacional)
RPC - Remote Procedure Call (Chamada remota de procedimento)
SOA – Service-Oriented Architecture (Arquitetura Orientada a Serviços)

SUMÁRIO

1 INTRODUÇÃO

Nos próximos anos, ante os eventos esportivos que acontecerão no Brasil em 2014 e 2016, a Copa do Mundo de Futebol e as Olimpíadas, respectivamente, está previsto a chegada de turistas de diferentes nacionalidades. Dentre os grandes problemas de infraestrutura das cidades brasileiras, um dos principais certamente é o transporte público. Tanto o turista bem como os moradores das cidades precisam de informações de melhor qualidade a respeito de qual meio de transporte utilizar e em que momento o mesmo estará disponível, mas isso não acontece atualmente na maioria das cidades.

As nossas cidades precisam de uma solução para viabilizar a comunicação dos serviços de transporte público a todas essas pessoas, de forma que, independentemente da língua e do lugar, essa informação deverá estar disponível. A grande barreira para implantação de um serviço como esse é que o modelo de organização do transporte no Brasil é descentralizado.

O Google possui um serviço chamado de Google Transit responsável por distribuir dados de rotas de transporte público e informações em tempo real, tais como a posição de um veículo e o estado de uma viagem programada. Através de um serviço desses será possível obter uma forma de centralizar todas as informações dos órgãos ou empresas responsáveis pelo transporte.

Mesmo tendo uma maneira de centralizar todos os dados relacionados aos serviços de transporte, existe ainda o desafio de como cada órgão ou empresa irá obter esses dados em tempo real. Além disso, como esses dados serão disponibilizados seguindo um modelo de dados de um serviço como o Google Transit.

1.1 Problema

Para disponibilizar dados de transporte público de forma centralizada ao público em geral, existem serviços como o Google Transit. Entretanto, para obter esses dados não existe uma forma simplificada, já que cada órgão ou empresa possui seu próprio sistema computacional. Somado a isso, dados em tempo real precisam ser obtidos diretamente de todos os veículos. Para organizações de pequeno porte pode tornar-se inviável, pelo fato de não possuírem recursos suficientes, ou seja, qualquer solução a ser proposta precisa apresentar uma significativa redução de custos.

O trabalho JORGE (2011) abordou o primeiro problema. Através do desenvolvimento de um sistema de cadastro de todos os órgãos e empresas responsáveis pelo transporte público, seguindo o modelo de dados do Google Transit, foi mostrado uma maneira de centralizar todos os serviços de transporte oferecidos por uma cidade em uma base de dados.

Em relação ao outro problema, de como obter dados em tempo real dos veículos, existem as seguintes questões: qual tecnologia utilizar para transmitir esses dados de um veículo até um servidor; que tipo de dispositivo e sistema será utilizado no veículo. Além disso, é necessário tornar esses dados disponíveis na internet seguindo um modelo de dados padronizado como apresentado pelo trabalho JORGE (2011).

Diante de alguns problemas apresentados no tocante aos aspectos relativos à mobilidade urbana, estamos propondo com este trabalho desenvolver uma solução para proporcionar, de uma maneira organizada, o fornecimento de serviços de transporte público, de sorte tal que no contexto de uma cidade, informações sobre os coletivos possam ser transmitidas, em tempo real, para o público em geral.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral do trabalho será o desenvolvimento de uma aplicação que permita a monitoração de informações associadas ao transporte público e compatíveis com o sistema Google Transit, com enfoque nas informações em tempo real.

1.2.2 Objetivos Específicos

Estudar as tecnologias mais atuais da Spring Source para desenvolvimento de aplicações Java, o Google Web Toolkit para desenvolvimento de aplicações web, e a integração dos dois.

Estudar como desenvolver uma aplicação para o sistema operacional Android.

Baseado no modelo de dados relacional apresentado em JORGE (2011), ampliar o mesmo para suportar dados em tempo real de uma organização de transporte público como, principalmente, a latitude e a longitude de um veículo.

Entregar uma aplicação servidor, produzida com as tecnologias estudadas, responsável por disponibilizar dados em tempo real no formato estabelecido pelo sistema Google Transit e receber dados em tempo real de veículos cadastrados nessa aplicação.

Entregar uma aplicação cliente da aplicação servidor, produzida para o sistema operacional Android, responsável por obter dados de um veículo como a latitude e longitude.

1.3 Justificativa

Um dos maiores problemas da maioria das cidades no Brasil é o transporte. Muitas pessoas dependem do transporte público diariamente e não possuem nenhum tipo de informação a respeito dos veículos em tempo real, ou seja, caso problemas aconteçam com esses últimos, as pessoas são obrigadas a esperar na parada de ônibus. Além disso, com os grandes eventos que o Brasil está recebendo e irá receber, os turistas são prejudicados, pois, devido a barreira linguística, possuem dificuldades em descobrir como chegar nos destinos desejados utilizando nossos serviços atuais de transporte público.

Através do desenvolvimento de uma solução integrada ao Google Transit, tanto os turistas, como os moradores de uma cidade conseguirão obter dados em tempo real através de seus dispositivos móveis ou computadores. Para que essas informações não sejam restritas aos usuários que possuem dispositivos móveis e computadores, as próprias organizações poderão prover meios físicos de mostrar a localização dos ônibus, por exemplo, através de telões e display instalados próximos a paradas de ônibus.

Um smartphone ou tablet, atualmente, possuem aplicativos e tecnologia necessários para obter dados em tempo real como a geolocalização de um veículo, além de proporcionar um meio do condutor do veículo comunicar pessoas a respeito e questões acerca de congestionamento, alertas sobre atraso e acidentes, por exemplo. Por isso, esse trabalho propõe o uso de um dispositivo com Android, sistema operacional aberto, como meio transmissor de dados em tempo real nos veículos.

1.4 Organização do trabalho

Além do capítulo introdutório, no capítulo dois, é mostrado o estudo feito em relação a tecnologias e técnicas para o desenvolvimento de uma aplicação web moderna em Java. O Spring Framework é investigado, bem como o Google Web Toolkit.

Em seguida, no capítulo três, é apresentado o projeto a ser desenvolvido nesse trabalho, ou seja, como os dados precisam ser modelados, quais métodos e materiais são utilizados para o desenvolvimento, além das ferramentas utilizadas para testar o sistema.

Já no capítulo quatro, é oferecido o detalhamento do desenvolvimento do sistema. No capítulo cinco, são apresentados testes executados para demonstrar o funcionamento da aplicação desenvolvida, e por fim, seguem-se as conclusões gerais, somadas a sugestões de trabalhos futuros.

2 ESTUDO DE TECNOLOGIAS PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB

Nesse capítulo será documentado o estudo feito de tecnologias da Spring Source e do Google Web Toolkit. Esses estudos irão auxiliar no entendimento de como desenvolver uma aplicação web moderna.

Foi determinado o estudo dos frameworks da Spring Source, já que essa empresa possui algumas das melhores tecnologias para auxiliar o desenvolvimento de aplicações Java. Já o estudo do Google Web Toolkit foi uma escolha do autor, pois é uma tecnologia nova que vem ganhando força e é desconhecida do mesmo.

2.1 Visão Geral

Atualmente, para um programador Java, existem diversas ferramentas de desenvolvimento ágil de aplicações web que são, por exemplo, geradoras de código ou frameworks, que é um conjunto de bibliotecas. Quando existe uma ideia e o próximo passo é tentar inseri-la em um sistema computacional, cria-se o grande desafio de escolher as melhores ferramentas disponíveis no momento. A escolha deve levar em consideração que o desenvolvimento precisa ser rápido, a manutenção ser simples, e que o sistema precisará suportar a possibilidade de expansão no futuro, além de oferecer os melhores recursos para os usuários.

Um dos desafios é que para o desenvolvimento de uma aplicação compatível com o modelo dos usuários atuais, existe a necessidade da mesma suportar serviços web para poder alimentar sistemas móveis, independente da tecnologia desses últimos.

Quando se pensa em iniciar um projeto web com Java, existem diversos fatores a considerar: qual framework utilizar, qual tecnologia do cliente web, qual a tecnologia dos serviços web, qual a tecnologia de segurança do sistema, onde o sistema será hospedado, qual será o meio de armazenamento utilizado, entre outros aspectos. Além desses, existe a questão de como unir todas essas opções em um sistema e configurá-las para que o mesmo funcione.

Durante uma pesquisa de tecnologias para criar aplicações web, foi identificado o Spring Roo como uma das ferramentas que mais ganhou força nos últimos anos, sendo tema, inclusive, durante a Google IO 2011 Brasil, Conferência de Desenvolvimento organizada pela Google, com o a criação de uma aplicação com o Google Web Toolkit.

O Spring Roo pode ser caracterizado, de acordo com seus criadores, como um gerador de aplicações, ou, como um gerador de código. Através de algumas linhas de definições e em aproximadamente 5 minutos, pode-se criar uma aplicação web em Java com acesso a banco de dados, páginas web, que permitem operações de salvar, atualizar, criar e remover elementos definidos naquelas configurações. Além de tudo já descrito, precisa ser dito que o Spring Roo suporta diversas tecnologias como o Google Web Toolkit, Aspect-J, Google App Engine, Adobe Flex, Hibernate, Spring framework, Spring Security, entre muitas outras. O Spring Roo parece ser a ferramenta perfeita para quem nunca criou uma aplicação Java web.

Para que um estudo do Spring Roo possa ser feito, primeiramente, deve-se entender o Google Web Toolkit.

2.2 Google Web Toolkit

GWT, ou Google Web Toolkit, é, segundo a Google, “um kit de ferramentas para desenvolvimento e para otimização de aplicações que são executadas em navegadores. O objetivo é permitir um desenvolvimento produtivo de aplicações web de alta performance, sem que o desenvolvedor precise ser um especialista em como os navegadores funcionam, XMLHttpRequest e Javascript.“

Segundo o livro “GWT in Action, segunda edição”, “GWT é uma caixa de ferramentas. Não é uma linguagem, um caminho ou um framework. É um conjunto de ferramentas que servem para fornecer um fácil caminho de escrever sofisticadas e confiáveis aplicações AJAX usando Java. GWT não foi feito para tirar vantagens do runtime do Java, pelo contrário, utiliza a linguagem Java e as ferramentas já existentes para essa.”

Na minha opinião, vejo o GWT como uma forma de programadores Java criarem aplicações que rodam em navegadores sem precisar estudar HTML e Javascript. E isso já é uma grande virtude do GWT, pois todo o esforço do desenvolvedor será na aplicação, ao invés de ter parte, ou de repente, a maior parte do esforço sendo investido no estudo de como cada navegador interpreta o Javascript. O GWT possui um mecanismo de gerar código Javascript específico para cada navegador, e o ultimo só irá obter o código mais otimizado para ele.

Além do fato da programação ser em Java, de gerar códigos específicos por navegador, outra vantagem do GWT é permitir diferentes implementações de interfaces com o usuário que são acionadas conforme condições: por exemplo, pode-se desenvolver uma interface gráfica mais simplificada que será ativada somente quando o GWT detectar um navegador de um dispositivo móvel. Nas

últimas versões da ferramenta, os desenvolvedores do GWT estão incorporando otimizações para melhorar o desenvolvimento de aplicações moveis.

Os desenvolvedores do GWT recomendam o uso da arquitetura MVP, ou “Model View Presenter” para a criação de aplicações GWT.

2.2.1 MVP no GWT

O MVP foi a arquitetura escolhida pelos desenvolvedores do GWT como o padrão ideal para desenvolvimento de aplicações GWT. Uma das grandes vantagens da utilização desse padrão é um desacoplamento dos três componentes: “Model”, “View” e “Presenter”. Isso irá facilitar o desenvolvimento, já que desenvolvedores podem, então, trabalhar simultaneamente no projeto, e testes podem ser feitos por componente.

Especificando sobre cada um dos três componentes do MVP, temos que o “Model”, ou, em português, “Modelo”, consiste o domínio da aplicação.

Já o “View” consiste em um componente de visualização que possui diversos outros componentes que contém os dados a serem exibidos. Um “View” não deve saber o que está exibindo, e sim somente que componentes possuem, como, por exemplo, uma caixa de texto, ou uma lista de caixas de texto.

E o “Presenter”, ou “Apresentador”, é aquele que possui a lógica da aplicação e irá determinar o que será exibido e em qual “View”. Além disso, terá a responsabilidade de aguardar e responder eventos produzidos no “View”.

O GWT versão 2.1 introduziu um framework chamado de “Activities and Places”. Esse permitirá a criação de URL únicos para cada parte da aplicação, e oferece um suporte para gerenciamento do histórico de navegação, permitindo o uso do botão “Voltar” do navegador pelo usuário. Essas duas questões do foco desse

novo framework estavam no topo da lista das desvantagens da utilização de GWT para construção de aplicações web. O framework foi feito para trabalhar em conjunto com o conceito de MVP.

Uma “Activity”, ou “Atividade” em português, representa o que o usuário está fazendo em uma determinada porção de uma pagina web. Um “Place” ou “Lugar” em português,é um objeto Java que representa um estado particular da interface gráfica, e o mesmo pode ser convertido para uma URL.

Unindo o MVP com o “Activities and Places” em GWT, obtém-se a arquitetura de uma aplicação moderna GWT, e é o padrão utilizado pela ferramenta Spring Roo para gerar aplicações GWT.

2.2.2 MVP e “Activities and Places”

Uma “View” nessa arquitetura é uma parte de uma interface gráfica que está associada com um Apresentador. Deve ser representada por uma interface Java, já que isso permitirá diferentes “Views” que serão ativadas conforme condições aplicadas na configuração da aplicação. Isso será utilizado para oferecer diferentes interfaces gráficas, baseado no dispositivo utilizando a aplicação.

É recomendado a utilização de um “ClientFactory”, que é responsável por guardar as “Views”, e, através do “ClientFactory”, pode-se obter a implementação correta para cada “View”, o que volta ao exemplo do caso do dispositivo móvel.

Uma Atividade deve implementar um Apresentador e deve, ao iniciar, obter uma “View” do “ClientFactory” e colocar a mesma na tela. Dessa forma, a Atividade não terá controle sob qual “View” deve ser criada para o contexto da aplicação, ou seja, a mesma Atividade pode ser utilizada para uma aplicação móvel ou uma

aplicação para um navegador usual. A “View” obtida deve receber o seu Apresentador, que, no caso, seria a própria “Atividade”.

Além disso, uma Atividade sempre receberá um Lugar, ou seja, esse último poderia ser dito como o responsável por ditar a lógica de negócio na Atividade. Por exemplo, um Lugar pode representar o número único de um funcionário, ou seja, toda vez que a Atividade for iniciada com esse Lugar, os dados do funcionário descrito serão buscados pela Atividade e exibidos pela “View”, que pode ser para smartphone ou para um navegador comum, obtida do “ClientFactory”.

Nessa nova arquitetura então, além de “Model”, “Viewer” e “Presenter”, temos os conceitos de “Activity” e “Places”. Até agora, falou-se apenas de uma “Activity” funcionando individualmente, mas, normalmente, existe uma comunicação entre as Atividades que estão ativas.

Essa comunicação se dá pelo “EventBus”, que é um canal de transmissão de eventos da aplicação. Quando ocorre um evento significativo para outros elementos da aplicação, uma Atividade deve mandar um evento pelo canal, e assim, todas as atividades que, ao iniciar seu ciclo de vida, registrarem que desejam receber eventos daquele tipo, irão recebê-lo, caso as mesmas estejam em execução. Um exemplo para esse processo é na hipótese de um usuário autenticar-se: atividades vão querer receber essa informação para poder personalizar os dados exibidos pelas mesmas.

Para troca de atividade, existe um objeto chamado “ActivityManager”, ou Gerenciador de Atividades, que necessita receber o “EventBus” e um outro objeto chamado de “ActivityMapper”, que é responsável por mapear um lugar para uma atividade. O Gerenciador de Atividades também precisa receber uma área da interface gráfica que ele deve atuar, ou seja, podem existir vários gerenciadores por aplicação. Por exemplo, o menu pode ter um gerenciador, enquanto o restante pode

ter outro gerenciador. Através de eventos, atividades de um gerenciador podem comunicar-se com atividades de outro gerenciador.

Já para solução do problema do botão “voltar”, ou seja, do histórico de navegação da aplicação, o GWT utiliza o “PlaceHistoryHandler” que provém um mapeamento bidirecional entre URL e o “Place”. Toda vez que o usuário clica no botão “voltar” do navegador, o “PlaceHistoryHandler” recebe um evento com um token, uma “string” que representa um lugar, e, a partir desse, manda um evento de mudança de Lugar para o “EventBus” através do “PlaceController”.

Em geral, o MVP com Atividades e Lugares produz uma aplicação com uma quantidade de código maior que uma arquitetura comum MVP. Por outro lado, irá oferecer muitas oportunidades de reuso de código, caso a aplicação seja de grande porte, além de oferecer o conceito de Lugar, que, para uma aplicação web, é extremamente importante.

O MVP oferece uma arquitetura para o desenvolvimento de lado do cliente web de uma aplicação web, mas não define absolutamente nada em relação a comunicação entre o cliente e o servidor, ou seja, o acesso ao “Model” ou Domínio da aplicação. A tecnologia “Request Factory”, ou Fábrica de Pedidos, também produzida pelos desenvolvedores do GWT, irá prover um mecanismo do cliente GWT acessar o domínio da aplicação, que, normalmente, está em um banco de dados em um servidor.

2.2.3 Request Factory

Também na versão 2.1 do GWT, foi introduzida uma alternativa interessante de comunicação entre o provedor de dados e o cliente web GWT chamada de “Request Factory”. O principal benefício da utilização desse mecanismo é que o

mesmo possui um controle de versão dos dados trafegados entre o cliente e o servidor, ou seja, um mecanismo de cache é feito, e além disso, apenas dados modificados entre o cliente e o servidor são transmitidos.

Outro fator positivo é que somente dados requisitados são enviados, ou seja, se um objeto A possui o objeto B, caso o cliente faça uma requisição para o servidor do objeto A, somente este será enviado. Apenas será enviado o Objeto B, se explicitamente pedido.

O aspecto negativo desse mecanismo é a utilização de duas vezes mais classes que um padrão clássico como o GWT-RPC, onde o próprio objeto do domínio era transmitido para o cliente: para cada entidade do domínio, precisa-se de uma interface do tipo “Proxy”, que é uma interface com os “getters” e “setters” da entidade, ou seja, métodos para pegar e modificar atributos e uma interface do tipo “Request”, que é uma interface que contém os métodos para pedidos de dados. Considerando-se a possibilidade de criar scripts para geração de código, isso pode acabar não sendo um problema tão grave.

Por fim, o aspecto mais negativo é a falta de uma validação dessas interfaces “Proxy” e “Request” em tempo de desenvolvimento. Somente utilizando a compilação da aplicação inteira e, caso a mesma não possua nenhum erro, essas classes são validadas. Além disso, existem outros problemas que a validação não cobre, como, por exemplo, a tentativa de modificação de um atributo em que o “setter” não está declarado no Proxy.

Em geral, acredito que essa tecnologia é interessante, principalmente devido a esse gerenciamento dos dados pelo lado do cliente. Hoje, cada vez mais, as aplicações estão consumindo muitos dados e um gerenciamento eficiente irá fazer diferença no tempo de carregamento das páginas.

2.3 Estudo Spring Roo

A missão do Spring Roo é “fundamentally and sustainably improve Java developer productivity without compromising engineering integrity or flexibility“, ou, em português, “melhorar profundamente a produtividade Java de forma sustentável sem comprometer a integridade ou a flexibilidade da arquitetura”. O Roo suporta diversas tecnologias importantes para os sistemas usuais em Java e um grande volume de escolha de opções, permitindo que o usuário basicamente monte sua arquitetura como quiser, ou pelo menos essa é a intenção.

Além de gerar a aplicação web, o Spring Roo faz manutenção dela: quando mudanças são feitas em classes Java sob seu controle, como, por exemplo, classes do domínio da aplicação, automaticamente, o Roo faz uma atualização na interface gráfica gerada por ele. Outra característica é a oportunidade de troca de tecnologia do provedor JPA por meio do Roo ou pelo próprio programador, já que é configurado de forma a não interferir no código. Java Persistence API é um framework padrão definido pelo JCP (Java Community Process) para gerenciamento do banco de dados e, por meio dele, pode-se fazer o mapeamento de classes Java para tabelas em um banco de dados relacional.

Uma das questões que se apresenta quando alguém se vê frente dessa ferramenta é se ela pode ser utilizada, de forma efetiva, para montar aplicações reais ou se ela deve ser usada apenas para fins de protótipos. Uma das formas de verificar é se utilizando do Roo, que vem com diversos exemplos demonstrando como utilizá-lo, para criar uma aplicação com diversas combinações de tecnologias.

2.3.1 Descrevendo uma Aplicação através do Roo

Antes da análise de um dos exemplos, necessário definir como gerar uma aplicação com o Roo. Este pode ser iniciado através de um terminal, por meio do executável do mesmo, ou pode ser iniciado por meio de uma extensão para o Eclipse, ferramenta de código aberta para desenvolvimento. Uma definição de aplicação para o Roo é realizada através de um arquivo de texto que possui uma sequencia de comandos a serem executados pelo Roo. Como alternativa, pode-se executar os comandos individualmente.

Observando exemplos de modelos de aplicações geradas pelo Spring Roo, pode-se perceber que a configuração de uma nova aplicação é extremamente fácil. “Expenses” é o nome de uma amostra que vem junto com a ferramenta, e este demonstra como criar uma aplicação simples com o provedor de banco de dados Hibernate e o Google Web Toolkit como cliente web da aplicação.

O primeiro comando, descrito abaixo, define o pacote do projeto, ou, em termos de Java, isso seria o “package” principal da aplicação. Essa definição é obrigatória em uma aplicação Roo.

project --topLevelPackage org.springsource.roo.extrack

Já o comando abaixo define o provedor de banco de dados do sistema e o sistema de banco de dados. No caso, o mesmo está definindo que o Hibernate irá prover os dados para o sistema, que serão armazenados no banco de dados Hypersonic, que é um banco de dados simples que pode ser rodado na memória do computador e é utilizada muito em protótipos e para testes.

jpa setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY

Em seguida, uma sequencia de comandos. Claramente pode-se observar que o primeiro comando irá criar uma classe nomeada “Gender” do tipo Enum, uma enumeração em Java. O que pode não ser tão claro é a utilização do termo “~” para dizer que o nome do pacote em que a classe deve ficar é construído a partir do pacote principal definido no primeiro comando dessa definição. Após o comando de criação da classe, pode-se observar a criação de dois membros da enumeração “Gender”, o “MALE” e o “FEMALE”. Uma observação que pode ser feita é que os dois comandos de criação de constantes da enumeração não possuem nenhuma referência ao tipo de enumeração “Gender”. Isso acontece porque o Roo assume que comandos para criação de elementos dentro de uma classe sempre são executados para a última classe definida.

enum type --class ~.shared.domain.Gender

enum constant --name MALE

enum constant --name FEMALE

Agora, pode-se observar a sequencia de comandos que irá criar a classe “Employee”. Nota-se que o primeiro comando é “entity jpa”, e isso está dizendo que a classe “Employee” será uma entidade JPA e será mapeada no banco de dados definido pela aplicação. Além disso, “entity jpa” irá produzir métodos para salvar, criar, procurar, atualizar e remover a entidade. A opção “--testAutomatically” está dizendo ao Roo que serão gerados testes automáticos para essa entidade. Isso significa que os métodos gerados para essa entidade serão testados.

Abaixo do primeiro comando, pode-se observar a criação dos atributos do “Employee”. Esses comandos são simples, sempre começando com “field” e em seguida o tipo do atributo como, por exemplo, “string” ou “boolean”.

Observando o terceiro comando da lista abaixo, percebe-se a presença de uma opção chamada de “--sizeMin 3”, outra chamada “--sizeMax 30”, e por último a “—notNull”. O Spring Roo permite a caracterização dos atributos de uma entidade JPA, ou seja, definir restrições daquele atributo como o tamanho mínimo e máximo, ou mesmo que não pode ser vazio.

O quarto comando abaixo “field reference –-type Employee supervisor”, é para criação de um relacionamento “Muitos-para-Um”, onde a opção “--type Employee” indica outra entidade JPA já definida que será o “Muitos”. Aqui, nesse exemplo, o comando está indicando que um “Employee” possui um “supervisor” do tipo “Employee”, e que o mesmo “supervisor” pode ser relacionado com outros “Employees”.

```
entity jpa --class ~.server.domain.Employee --testAutomatically
field string --fieldName displayName --notNull
field string --fieldName userName --sizeMin 3 --sizeMax 30 --notNull
field string --fieldName department
field reference --type Employee supervisor
field enum --fieldName gender --type ~.shared.domain.Gender
field boolean --fieldName admin –notNull
```

Abaixo, está descrito os comandos para criação da entidade “Report” e o único aspecto que pode ser considerado novo em relação aos comandos anteriores

é a presença da opção “--type” para o comando “field”, no quarto comando: pode-se definir o tipo de um atributo diretamente para a classe do mesmo.

entity jpa --class ~.server.domain.Report --testAutomatically

field string --fieldName purpose

field string --fieldName notes

field date --fieldName created --type java.util.Date

field string --fieldName department

field reference --type Employee reporter

field reference --type Employee approvedSupervisor

Finalizando a configuração das entidades utilizadas pela aplicação, estão abaixo os comandos de criação da entidade “Expense”. Talvez a única novidade aqui é o comando “field number”: esse comando permitirá uma maior customização de atributos relacionados a valores numéricos, por meio de opções como “--digitsInteger”, que descreve o número máximo de dígitos da parte inteira do número, ou “—digitsFraction”, que descreve o número máximo de dígitos da parte fracionária de um número, entre muitas outras customizações descritas na documentação do Roo.

entity jpa --class ~.server.domain.Expense --testAutomatically

field number --type java.lang.Double amount

field string --fieldName description

field reference --type Report report

field string --fieldName approval

field string --fieldName category

field date --fieldName created --type java.util.Date

field string --fieldName reasonDenied

Depois de configurar a parte do domínio da aplicação “Expenses”, precisa-se configurar a parte do cliente web, que, no caso desse exemplo, é o Google Web Toolkit, ou mais conhecido como GWT. O comando abaixo irá configurar as bibliotecas necessárias para que o GWT esteja apto a ser usado nesse projeto.

web gwt setup

O último comando dessa aplicação irá gerar todas as classes necessárias para uma aplicação GWT completa, seguindo o padrão MVP.

web gwt all --proxyPackage ~.client.proxy --requestPackage ~.client.request

Depois de feita essa análise de como descrever uma aplicação seguindo a sintaxe do Spring Roo, o próximo passo é gerar e verificar a aplicação que o Roo gerou.

2.3.2 Gerando e analisando a Aplicação

Após a análise do arquivo de configuração do projeto do Spring Roo, o próximo passo é a geração do projeto Java. Através do comando abaixo, estando com a aplicação do Roo em execução, todos os comandos descritos no arquivo “expenses.roo” serão executados em sequencia.

script --file expenses.roo

Em seguida, o projeto pode ser importado, por exemplo, no STS (SpringSource Tool Suite), que é baseado no Eclipse, ou pode ser importado no Eclipse, caso o plugin do STS esteja instalado, já que o Spring Roo irá trabalhar caso modificações sejam feitas nas classes do domínio.

2.3.2.1 Rodando a aplicação

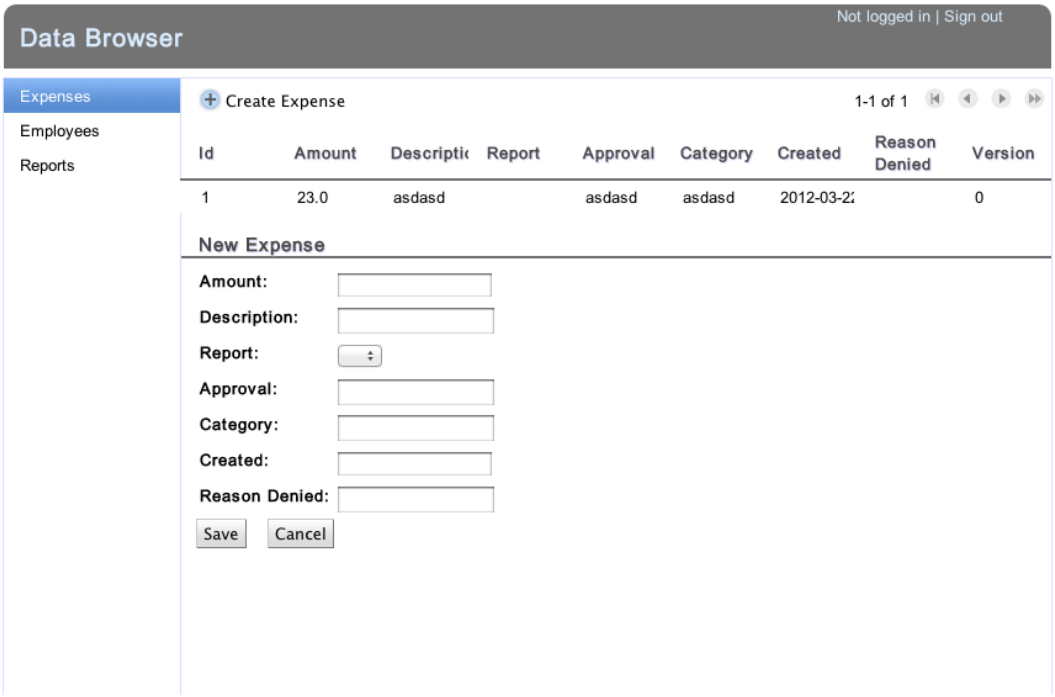
Para rodar a aplicação, é necessário possuir o Maven. O Apache Maven é um programa que, através de um arquivo único de configuração, consegue gerenciar todas as dependências do projeto, todos os ciclos de construção do projeto, entre outras funções. Através de um arquivo “pom.xml”, um projeto pode ser descrito de forma que o Maven irá executar todas as operações definidas e montar a aplicação.

No caso do Spring Roo, ele gerou o arquivo “pom.xml”, onde definiu todas as bibliotecas necessárias para a execução do projeto, como, por exemplo, as bibliotecas do GWT e do Hibernate, e configurou, também, que a validação das interfaces “Proxy” e “Request” do mecanismo “Request Factory” deve ser rodada ao construir a aplicação web, bem como instruções de como montar a aplicação.

Através do comando “mvn gwt:run”, o Maven irá compilar, validar, construir a aplicação e executar a mesma.

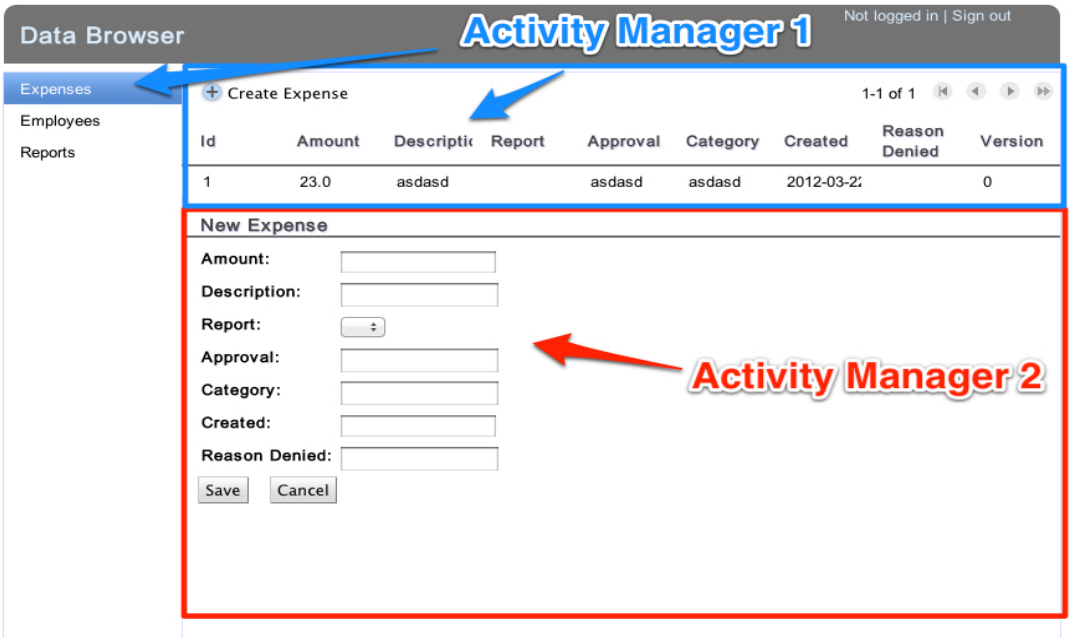
Como, pode-se verificar na figura 1, o Roo gera uma aplicação genérica, onde, no menu estão todas as classes do domínio da aplicação, e a direita é sempre a lista dos dados no banco de dados do tipo do domínio selecionado no menu.

Figura 1 - Exemplo de aplicação gerada pelo Spring Roo



Abaixo, na figura 2, pode-se observar como o Spring Roo organiza a aplicação em termos de GWT. Em azul está a área influenciada pelo “Activity Manager 1”, enquanto em vermelho a área influenciada pelo “Activity Manager 2”. A área em azul é diretamente influenciada pelo menu selecionado. Já a área em vermelho, só é acionada quando existe uma chamada para um “Place” dos detalhes de algum objeto do domínio selecionado ou quando um novo deve ser criado.

Figura 2 -Exemplo detalhado de aplicação gerada pelo Spring Roo



2.3.2.2 Analisando o código gerado

As entidades JPA como, por exemplo, a “Employee”, são controladas por anotações que referenciam aspectos. Além dessas, pode-se observar as anotações que caracterizam os atributos, como, por exemplo, “@NotNull” ou “@Size”. Essas anotações são importantes, pois estabelecem também a configuração no banco de dados de cada coluna que representará o atributo.

Já a anotação “@RooJavaBean” mapeia aquela entidade a um arquivo gerenciado somente pelo Roo, onde fará os “getters” e “setters” de cada entidade. Isso simplesmente injetará os métodos na classe “Employee” durante a compilação.

Também a anotação “@RooJpaActiveRecord irá mapear para arquivos gerenciados pelo Roo que irão gerenciar aspectos como “Jpa_Entity”, responsável por elementos de entidades JPA como, por exemplo, o ID, e “Jpa_ActiveRecord”, responsável por operações como salvar, atualizar, remover, procurar e contar as entidades.

Em relação ao código gerado para transmissão dos dados do domínio da aplicação ao cliente web GWT, outro aspecto interessante do Spring Roo é que o mesmo gerencia as interfaces “Proxy” e “Request” do “Request Factory”. Ou seja, modificações de atributos também serão refletidos nessas interfaces. As interfaces “Proxy” são aquelas que estendem “EntityProxy”, enquanto as interfaces “Request” são aquelas que estendem “RequestContext”.

Um dos grandes problemas do Spring Roo é que o mesmo gera uma quantidade enorme de classes. No total, são geradas oito classes para fazer o trabalho das duas atividades, para cada entidade do domínio da aplicação. Uma das atividades deve listar todos os elementos daquela classe e a outra fazer a edição de um elemento ou a criação de um novo. Além de todas essas classes, ainda existem,

por classe de domínio, mais oito classes para o “View”, e ainda quatro arquivos do “UI Binder”.

Em muitas situações, nem todos os objetos de domínio devem ter essa função de serem completamente editáveis e acessíveis diretamente pelo usuário do sistema, e também, nem todos atributos deveriam estar disponíveis para edição. Existe uma classe responsável por mapear os atributos de entidades, como a “Employee”, para componentes de interface de usuário do GWT. Atributos que são do tipo “String” são mapeados para componentes gráficos do tipo “TextBox”, ou Caixa de Texto, enquanto atributos que são objetos, são mapeados para uma lista de valores, chamada de “ValueListBox”.

2.3.3 Conclusões sobre Spring Roo.

A grande vantagem da utilização do Spring Roo é que, em 5 minutos, aproximadamente, pode-se gerar uma aplicação web funcional e plenamente configurada. E isso por si só, é extraordinário, pois através da criação de uma aplicação com Spring framework, GWT e Hibernate configurados, são economizados dias de trabalho, principalmente, para pessoas sem experiência em desenvolvimento web.

As desvantagens são: a necessidade de utilização de um plug-in no Eclipse; ter o Roo rodando a todo momento durante o desenvolvimento; e a geração de classes em excesso no caso do cliente GWT. Em compensação, o Roo suporta uma remoção própria e completa. Isso significa que o Roo pode ser utilizado apenas como gerador de código inicial e estrutura completa do projeto.

Uma outra nota sobre o Roo em relação ao GWT, é que o código é confuso para uma pessoa sem experiências com GWT, como o Autor. A quantidade

excessiva de classes não parece fazer muito sentido, pois, se por um lado, o Roo tentou generalizar, ao máximo possível, para evitar a geração de código repetido, por outro, mesmo assim, gerou código em excesso, código espalhado e de difícil compreensão, o que dificulta e muito a customização do mesmo.

Concluindo, acredito que o Roo deve, sim, ser utilizado como ponto de partida em um projeto Java web, já que os benefícios de construção inicial do projeto são favoráveis, quais sejam: ter o projeto e todas as suas bibliotecas configuradas, todas as classes de domínio mapeadas como JPA e, no caso do GWT, já ter todos os “Proxy” e “Request” de cada classe de domínio produzidos. -

2.4 Arquitetura de Software: SOA

Para desenvolvimento de uma aplicação web de médio a grande porte, precisa-se definir uma arquitetura. Considerando-se que, no caso desse trabalho, planeja-se o desenvolvimento de um cliente remoto em um dispositivo móvel, uma das melhores maneiras de criar aplicações com diversos clientes funcionando de forma independente é seguir a arquitetura SOA, ou Service-oriented Architecture.

Segundo o Wikipedia, SOA é “um conjunto de princípios e metodologias para modelar e desenvolver software na forma de serviços que funcionam juntos. Esses serviços são funcionalidades de negócio bem definidas que são construídos como componentes do software que podem ser reutilizados para diferentes propósitos“.

A grande vantagem da utilização de SOA no caso de uma aplicação web é deixar a lógica de negócio centralizada e oferecer acessos para os diferentes clientes, como por exemplo, serviços web, usados por uma aplicação Android, ou a “Request Factory”, usada pelo GWT. Isso irá proporcionar uma fácil manutenção e

uma oportunidade de crescimento dos clientes sem interferência nos outros já existentes.

Além disso, pode-se criar testes automatizados para essa camada dos serviços que são independentes dos clientes.

Possuindo uma arquitetura, o próximo passo é definir uma tecnologia que ofereça facilidades para desenvolver SOA. Como definido que um dos objetivos do trabalho é o estudo das tecnologias da Spring Source, principalmente, por serem utilizadas em grandes projetos por todo o mundo, será estudado o Spring Framework e como ele pode ajudar a montar uma estrutura SOA.

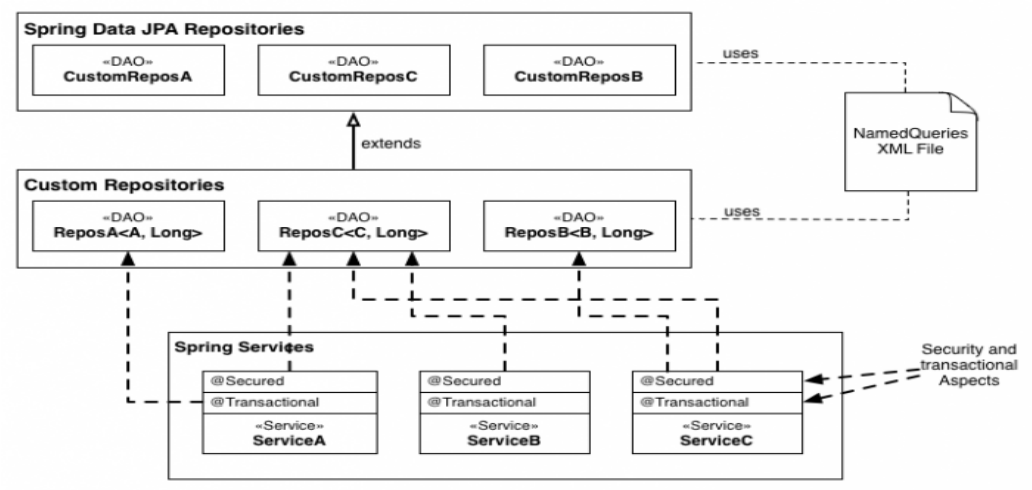
2.4.1 SOA com Spring Framework

O Spring Framework é um framework para desenvolvimento de aplicações Java. Fornece um conjunto de bibliotecas para proporcionar um desenvolvimento ágil de grandes aplicações. Dentre todas as características desse, pode-se destacar “Inversion of Control”, que é quando o ciclo de vida de objetos e suas dependências são administrados pelo Spring, “Authentication and Authorization”. O Spring possui o próprio framework para controle de acesso chamado de Spring Security, e “Data Access”, que é o módulo responsável por acessar o banco de dados.

O Spring framework oferece um grande suporte para SOA. Através da definição das classes Java com anotação “@Service”, o próprio Spring fica encarregado de fazer manutenção no ciclo de vida do objeto.

Abaixo, na figura 3, está um diagrama retirado de um blog [16] que mostra como SOA poderia ser aplicado utilizando recursos do Spring Framework.

Figura 3 - Exemplo de arquitetura SOA com Spring Framework



Fonte: http://crazygui.wordpress.com/2011/12/06/spring-gwt-software-architecture-for-scalable-applications-part-1

“Spring Data JPA Repository” são componentes gerados pelo Spring que têm como objetivo fornecer as operações básicas de acesso ao banco de dados a uma entidade JPA específica. Esses repositórios serão o canal de comunicação entre o serviço e os dados armazenados no banco de dados. Para definir um, no Spring Framework, basta criar uma interface Java com o nome desejado e que estenda a interface “JpaRepository”.

“Custom Repository” é um repositório JPA customizado, ou seja, que contém operações não definidas pela interface “JpaRepository” e no caso, é um componente opcional e deve estender essa interface.

Spring Services é a camada que contém todos os serviços da arquitetura SOA, responsáveis pela lógica de negócio da aplicação. Os serviços serão alimentados e irão alimentar os repositórios com os dados.

O elemento representado pelo nome “NamedQueries XML File” é um arquivo que contém operações com o banco de dados, que seria utilizado pelos repositórios customizados. Já que um dos objetivos é utilizar as melhores tecnologias disponíveis, foi lido no blog do Spring Source, a respeito de uma nova tecnologia para criação de buscas em banco de dados que é a QueryDSL.

A união desses elementos vai formar a camada de serviço da aplicação. O objetivo da mesma é prover dados para o cliente web, para os serviços web e para futuras extensões da aplicação. Como já mencionado, o principal benefício de possuir essa camada é a centralização da entrada e da saída dos dados, o que facilita a solução de problemas em relação a segurança, a consistência e a distribuição de dados.

2.4.2 QueryDSL

Segundo o próprio site, “QueryDSL é um framework que permite a construção de queries type-safe parecidas com SQL para diversas back-ends como JPA, JDO e SQL em Java.”

Um dos aspectos interessantes desse framework é a compatibilidade com o “JPA 2 Criteria API”, que é um padrão definido pelo “Java Community Program” para mapear objetos em banco de dados relacional. A vantagem desse padrão é não permitir que queries inválidas sejam feitas e é suportado pelos principais provedores de banco de dados como o Hibernate. Interessante é que um repositório no Spring suporta buscas por meio do “JPA 2 Criteria API”, e já que o “QueryDSL” também

permite tais buscas, pode-se utilizar esse framework sem problemas de compatibilidade.

A maior vantagem de utilizar o QueryDSL é a facilidade do entendimento do código necessário para fazer buscas no banco. Oliver Gierke, no blog do Spring Source [19], mostrou as diferenças entre o framework padrão JPA Criteria e o QueryDSL. Oliver propôs, como cenário de teste, no caso de uso, a partir dos consumidores que são fiéis de um sistema, ou seja, que estão há bastante tempo cadastrados.

Abaixo está o código, caso, o JPA Criteria fosse utilizado. Pode-se notar que não é um código amigável.

```
LocalDate today = new LocalDate();

CriteriaBuilder builder = em.getCriteriaBuilder();
CriteriaQuery<Customer> query = builder.createQuery(Customer.class);
Root<Customer> root = query.from(Customer.class);

Predicate hasBirthday = builder.equal(root.get(Customer_.birthday), today);
Predicate isLongTermCustomer =
```

```
builder.lessThan(root.get(Customer_.createdAt), today.minusYears(2);
        query.where(builder.and(hasBirthday,
isLongTermCustomer));
        em.createQuery(query.select(root)).getRes
ultList();
```

Além do código ser inadequado, outro problema foi identificado pelo Oliver: os predicados como “fazAniversário” ou “éAntigoCliente” não podem ser reutilizados. Para resolver esse problema, a equipe do Spring Source, criou uma interface “Specification”. Abaixo está um exemplo para esse caso estudado de definição de cada um dos predicados, de forma que os mesmos podem ser reutilizados.

```
        public
CustomerSpecifications {

         public static Specification<Customer>
customerHasBirthday() {
           return new
Specification<Customer> {
             public Predicate toPredicate(Root<T> root, CriteriaQuery query,
CriteriaBuilder cb) {
               return
cb.equal(root.get(Customer_.birthday), today);
```

```
    }

  };

}

        public static Specification<Customer>
isLongTermCustomer() {
            return new
Specification<Customer> {
              public Predicate toPredicate(Root<T> root, CriteriaQuery query,
CriteriaBuilder cb) {
                return cb.lessThan(root.get(Customer_.createdAt), new
LocalDate.minusYears(2));

    }

  };

}
```

E abaixo, uma forma de utilizar os predicados num repositório responsável por trazer esses consumidores.

```
customerRepository.findAll(where(customerHasBirthday()).and(isLongTermC
onsumer)));
```

Essa solução é interessante em relação a JPA Criteria, pois permite o reuso de código, mas ainda não é a ideal. Por isso, um grupo criou um projeto aberto chamado de QueryDSL, com o objetivo de tornar a codificação desses predicados mais amigável. Abaixo, pode-se observar um exemplo com essa tecnologia.

```
BooleanExpression customerHasBirthday = customer.birthday.eq(today);
BooleanExpression isLongTermCustomer =
customer.createdAt.lt(today.minusYears(2));
customerRepository.findAll(customerHasBirthday.and(isLongTermCustomer)
);
```

O resultado acima é um código que qualquer programador consegue entender e que facilmente pode ser codificado de forma a torná-lo reutilizável. Define-se uma expressão booleana que representa que o cliente faz aniversário hoje, enquanto a outra expressão representa que o cliente foi criado antes de dois anos atrás.

O único inconveniente dessa tecnologia é a necessidade da geração de classes específicas para cada entidade JPA. Porém, isso não será algo preocupante, devido a fácil integração com o Maven. Através de adição de um plug-in na definição do arquivo “pom.xml”, as classes JPA são detectadas e as correspondentes são geradas. Por fim, essas novas classes podem ser usadas para

criação dos predicados, e, no exemplo acima, o objeto de uma dessas classes seria o “customer”.

2.4.3 Integrando GWT com SOA

Até agora foi mostrado o que é a arquitetura SOA, como ela é estruturada e como ela pode ser feita com o Spring Framework. O próximo passo é identificar como o cliente GWT irá ser conectado a essa.

O cliente GWT, toda vez que precisar de dados, ele irá fazer uma chamada assíncrona para a camada de serviço.

Quando um usuário entra em uma URL de um cliente GWT ou quando ele é redirecionado para um Lugar, normalmente, uma nova Atividade é criada, que é um Apresentador. A primeira coisa que a Atividade faz é colocar uma “View” obtida do “ClientFactory” na tela. Um segundo passo é obter dados junto ao serviço: para isso, através do mecanismo “RequestFactory”, é obtido um objeto do tipo “RequestContext” e uma chamada assíncrona é feita. A resposta, então, virá ou não, da camada de serviço através de um objeto do tipo “Proxy”, que pode ser chamado de um “data transfer object”, ou seja, um objeto que carrega só dados.

A aplicação GWT conhece apenas a “RequestFactory” e aí está um dos motivos da possibilidade de otimizações operadas por essa fábrica. Quando uma chamada assíncrona é feita, objetos que já existem no cache da “RequestFactory” irão retornar, ao invés da camada de serviço ser acionada.

2.4.4 Integrando clientes remotos com SOA: REST

A facilidade de criar uma aplicação web utilizando a arquitetura SOA é a possibilidade de oferecer os mesmos serviços que um cliente web oferece em dispositivos remotos como um dispositivo móvel de forma rápida, já que toda lógica de negócio está em uma camada especial de serviço.

Para comunicação de uma aplicação web com um dispositivo remoto, a maneira mais utilizada atualmente é através de serviços web seguindo a arquitetura REST.

REST, ou “Representional State Transfer” é uma arquitetura utilizada para comunicação distribuída na web. REST é extremamente mais simples que as outras tecnologias de serviços web e, por isso, está ganhando cada vez mais espaço.

Utiliza o próprio HTTP como protocolo, ou seja, não adiciona nenhuma camada extra que outras arquiteturas normalmente fazem. Um serviço web é considerado “RESTful” se ele adota as seguintes características: cliente-servidor, sem estado, ou seja, cada pedido tem um ciclo de vida curto e começa quando o pedido é feito e termina quando ele é respondido, entre outras características.

Através do vocabulário do HTTP, o REST descreve seus recursos. Através do método do HTTP, ou seja, “GET, PUT, DELETE, POST...” e o caminho do pedido, definem-se operações específicas. Enquanto o “status” do HTTP pode ser utilizado para definir mensagens do estado da resposta, por exemplo, se o pedido foi negado, ou se o pedido ocorreu sem problemas.

O Spring framework oferece suporte para disponibilizar serviços REST. Através de simples anotações, pode-se definir serviços web. Para que seja definido um serviço REST no Spring, basta anotar a classe como “@RequestMapping(/recurso)” e “@Controller”, ou seja, todos os pedidos feitos para “/recurso”, serão mapeados para essa classe. Os métodos da classe também

possuirão a anotação “@RequestMapping”, mas especificando o tipo de método HTTP que irão responder.

2.4.5 Conclusões sobre SOA

Utilizando a SOA, podemos concentrar todo a lógica importante da aplicação em um local único. Isso vai permitir testes simples, mas efetivos e que não envolvem lógicas especificas de clientes, e uma fácil manutenção da parte mais valiosa do sistema. Além disso, e talvez o fator mais importante, é que para o controle de acesso e escrita dos dados existirá somente uma entrada, a camada de serviço, o que irá facilitar configurações de segurança.

2.5 Segurança da Aplicação

Uma das grandes vantagens da utilização de um framework como Spring, é que ele possui diversos outros frameworks dentro dele totalmente compatíveis. Esse é o caso do Spring Security. Esse framework de segurança existe desde 2003, mas naquela época era nomeado Acegi Security. Foi incorporado ao Spring logo em seguida.

Montando uma aplicação web, a segurança é uma das partes principais da aplicação, já que o objetivo dessa aplicação é lidar com dados de usuários e restringir certas funções da aplicação, baseado nos direitos de acesso de cada usuário. Hoje em dia, não existe a ideia de inventar um sistema de segurança

somente para sua aplicação web Java, considerando que o Spring Security possui quase 10 anos de experiência e suporte para customização.

O grande problema é que não existe um passo a passo de como integrar o Spring Security numa aplicação GWT 2.4 (versão mais atual, utilizada pelo Roo), ou pelo menos, o autor não encontrou até esse momento. Existem diversos artigos e relatos, todos com seus méritos, mas nada que pudesse solucionar de forma fácil e eficaz todos os problemas.

Os próximos subcapítulos irão descrever alternativas de como integrar o Spring Security numa aplicação GWT.

2.5.1 Desafios de Segurança no GWT

O primeiro grande obstáculo de uma aplicação GWT é o conceito de que todo o código Java do cliente web será convertido para Javascript e será obtido pelo navegador do usuário que estiver utilizando a aplicação. Caso o código Javascript seja modificado, em teoria, o usuário poderia ter acesso a um recurso protegido. Um programador que fez a pergunta em [18] a respeito da segurança do código dele.

Esse código seria responsável por filtrar, toda vez que o usuário entrar em uma URL diferente da aplicação GWT, o Lugar gerado a partir do TOKEN da URL. O grande problema é que o código poderia ser modificado em um ataque, e o método, que pergunta se o usuário está autenticado, retornaria verdade sempre.

Então um dos problemas é que todos os recursos do sistema precisarão ser protegidos em dois níveis: nível de cliente, ou seja, o código acima seria o suficiente, e em nível de servidor, ou seja, toda vez que um usuário acessar um recurso, o próprio servidor irá verificar se aquele usuário possui direitos de acesso aquele

recurso. Mesmo que o cliente GWT seja modificado, os dados que retornam do servidor estarão protegidos.

2.5.2 Configurando o Spring Security

Nesse capítulo será mostrado como o Spring Security pode ser configurado em um projeto com a tecnologia GWT.

2.5.2.1 Configurando o Maven

O primeiro passo para a configuração do Spring Security, é através do Maven: deve-se adicionar as dependências desse framework. Um problema que acontece nesse momento é o conflito das dependências do projeto. Muitas vezes, bibliotecas diferentes utilizam versões diferentes de outras bibliotecas como suas dependências.

Uma maneira de verificar esse possível problema, é através do plug-in M2ECLIPSE, caso o Eclipse esteja sendo utilizado como ambiente de desenvolvimento. O M2ECLIPSE oferece possibilidade de buscas por todas as dependências, o que será muito útil para encontrar os culpados por conflitos de versões. Identificando os culpados, pode-se utilizar o Maven para excluir sub-dependências de dependências do projeto.

2.5.2.2 Entendendo as configurações "ApplicationContext"

Em uma aplicação com Spring Framework, pode-se notar diversos arquivos no formato XML chamados de "ApplicationContext". Esses arquivos são utilizados

para configuração de detalhes do Spring framework como um todo. Para poder configurar a segurança da aplicação, é necessário entender como esses arquivos funcionam.

O primeiro detalhe presente em arquivos desse tipo, é a presença da definição “beans”. Isso irá descrever quais opções de configurações estarão disponíveis no arquivo. Abaixo, na figura 4, é mostrado como está definido isso no arquivo “ApplicationContext.xml”, que pode ser considerado o arquivo de configuração principal da aplicação.

Figura 4 - Exemplo do arquivo de configuração do Spring Framework

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:util="http://www.springframework.org/schema/util" xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
     http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
          http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd
             http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
                   http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.1.xsd
                    http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd
            http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
           http://www.springframework.org/schema/util  http://www.springframework.org/schema/util/spring-util-3.1.xsd
            ">
```

Na primeira linha da definição “beans”, através da configuração “xmlns= .../schema/beans” está dizendo que sem nenhum “namespace”, pode-se acessar as configurações definidas no endereço “.../schema/beans”. Isso quer dizer que para criar um “bean”, que é uma definição especificada nesse endereço, pode-se definir diretamente como “<bean>”. Já numa definição especificada no “.../schema/util”, observada na segunda linha, deve-se utilizar “<util:opção_desejada>”.

Além dessas duas, outras definições de “namespace” são feitas, e por fim, é feito um mapeamento da definição de configuração e do arquivo “.xsd”, utilizado com a opção “xsi:schemaLocation”. Isso é feito para dizer de forma específica qual

versão de cada componente do Spring será utilizada e deve ser correspondente a versão definida das dependências determinadas no Maven.

Depois de entender como são definidos os caminhos das configurações, precisa-se analisar as configurações. Abaixo, pode-se observar duas configurações e a respeito do “namespace” chamado “context”.

A primeira definição exibida abaixo, na figura 5, está dizendo que as configurações nas classes Java, por meio de algumas anotações, serão levadas em consideração pelo Spring, como, por exemplo “@PostConstruct”, que indicaria que um método será executado logo depois de um objeto ser construído. Já a segunda configuração, diz que propriedades serão carregadas de todos os arquivos no formato “.properties”, no caminho “.../META-INF/spring/”, enquanto a terceira configuração diz que classes com anotações como “@Service”, “@Repository” e “@Component” poderão ser utilizadas como dependências em outras classes da aplicação, por meio, por exemplo, da anotação “@Autowired”. Isso será útil para, por exemplo, um objeto do serviço de autenticação, poder ter acesso ao objeto do repositório dos dados dos usuários.

Figura 5 - Exemplo do arquivo de configuração do Spring Framework

```
<context:annotation-config />

<context:property-placeholder location="classpath*:META-INF/spring/*.properties" />

<context:spring-configured />
```

Já na figura 6, está a configuração do acesso ao banco de dados. O “bean” da classe “BasicDataSource” será inicializado com as propriedades descritas no arquivo “database.properties”, utilizado na criação do “bean” da classe “LocalContainerEntityManagerFactoryBean”. Esse último “bean” será utilizado para

prover aos repositórios da aplicação o acesso ao banco de dados. Isso tudo será feito de forma automática, sem nenhuma outra configuração. Outro detalhe é que esse “bean” utilizará configurações do arquivo “persistence.xml” para obter o seu provedor de acesso ao banco de dados.

As grandes vantagens dessas configurações abaixo é que o banco de dados e o provedor de banco de dados são configurados através de arquivos simples e fora da configuração principal da aplicação. Isso vai permitir, por exemplo, que, desenvolvedores utilizem configurações de banco de dados diferentes, dependendo de onde a aplicação estiver sendo rodada. Para testes, pode-se utilizar um banco de dados em memória, enquanto, para aplicação real o MySQL por exemplo.

Figura 6 - Exemplo do arquivo de configuração do Spring Framework

```
<bean class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close" id="dataSource">
    <property name="driverClassName" value="${database.driverClassName}" />
    <property name="url" value="${database.url}" />
    <property name="username" value="${database.username}" />
    <property name="password" value="${database.password}" />
    <property name="testOnBorrow" value="true" />
    <property name="testOnReturn" value="true" />
    <property name="testWhileIdle" value="true" />
    <property name="timeBetweenEvictionRunsMillis" value="1800000" />
    <property name="numTestsPerEvictionRun" value="3" />
    <property name="minEvictableIdleTimeMillis" value="1800000" />
</bean>
<bean class="org.springframework.orm.jpa.JpaTransactionManager"
    id="transactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
<tx:annotation-driven mode="aspectj"
    transaction-manager="transactionManager" />
<bean
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
    id="entityManagerFactory">
    <property name="persistenceUnitName" value="persistenceUnit" />
    <property name="dataSource" ref="dataSource" />
</bean>
```

Na figura 7, está a definição do arquivo “persistence.xml”. Aqui, está definido que o provedor do banco será o “HibernatePersisente”, e todas suas propriedades

de configurações. Caso, por algum motivo, haja necessidade de mudança do provedor, basta a mudança desse arquivo apenas.

Figura 7 - Exemplo do arquivo de configuração do Spring Framework

```
<persistence-unit name="persistenceUnit" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.ejb.naming_strategy" value="org.hibernate.cfg.ImprovedNamingStrategy"/>
            <property name="hibernate.connection.charSet" value="UTF-8"/>
        </properties>
    </persistence-unit>
</persistence>
```

Uma aplicação utilizando o Spring Framework poderá possuir diversos "ApplicationContext". Por exemplo, através do arquivo "ApplicationContext-jpa", pode-se definir configurações para criação de repositórios baseado nas definições em interfaces java. Uma interface de um repositório de uma entidade JPA qualquer chamada de "AppUser" pode ser definida. Através da anotação "@Repository", e da extensão da classe "JpaRepository", isso seria o suficiente para a definição de um repositório no Spring. Outro detalhe, é a presença do "QueryDslPredicateExecutor", que é uma interface do Spring para compatibilidade com a biblioteca "QueryDSL".

As definições do "ApplicationContext-jpa" são simples: apontam para o pacote onde as interfaces dos repositórios são definidos. Um exemplo pode ser observado na figura 8, onde todas as interfaces definidas no pacote "br.ufsc.inf.teste.repository" seriam criadas e controladas pelo Spring.

Figura 8 - Exemplo do arquivo de configuração do Spring Framework

```
<repositories base-package="br.ufsc.inf.teste.repository" />
```

Para que esses arquivos de configuração sejam inicializados, existe a necessidade do carregamento dos mesmos pelo "Servlet" do Spring, que é uma classe responsável por processar os pedidos e as respostas para a aplicação. Isso é

feito no arquivo “web.xml”: através de um parâmetro de contexto, pode ser configurado que todos os arquivos, que iniciam com “ApplicationContext”, serão carregados pelo “DispatcherServlet” do Spring, que é responsável por todos os pedidos HTTP.

Para definir então, as configurações de segurança, basta a criação de um arquivo com o nome “ApplicationContext-security”, e o mesmo será carregado pelo Servlet.

2.5.2.3 Configurações de Segurança

Para definir as configurações de segurança, necessário, primeiramente, adicionar o “namespace” chamado de “security” para o novo arquivo “ApplicationContext-security.xml”.

Aplicações GWT possuem o desafio de serem codificadas completamente em Java, então, para fazer o processo de autenticação, não se pode utilizar os mecanismos como ter uma página com um formulário que faz um HTTP Post no endereço específico para tentar a autenticação. Isso é verdade em partes.

Uma alternativa interessante para proteger uma aplicação GWT seria a criação de uma página completamente independente da aplicação GWT, que faria o processo de autenticação com um formulário, e, caso fosse um sucesso, redirecionaria para a aplicação GWT. Já a outra alternativa é fazer o processo de autenticação totalmente por GWT.

2.5.2.3.1 Autenticação por Página

Para a primeira alternativa, com a página, a configuração no arquivo “ApplicationContext-security” seria da seguinte forma, observado em um exemplo chamado de “Spring-Social Showcase” da própria Spring Source em [20] na figura 17.

Através da configuração “http”, serão definidos os caminhos da aplicação protegidos. A configuração “use-expressions” irá dizer que expressões poderão ser utilizadas para condicionar acessos a diferentes caminhos da aplicação. Já a “create-session” irá informar se sessão deve ou não ser criada ao acessar os caminhos definidos a partir do “pattern”, ou padrão, que, no caso abaixo, seria “/**”, ou seja, toda aplicação. A opção “entry-point-ref” aponta para a definição do ponto de entrada da autenticação da aplicação.

Após essas definições iniciais, são especificadas as configurações de “logout”, mapeada para a URL “/logout”, ou seja, quando o usuário acessar essa URL, ele terá sua autenticação invalidada. Em seguida, pode-se observar padrões de URL que serão interceptados e a opção de acesso. O último padrão diz que a aplicação inteira “/**” precisa do acesso do tipo “está autenticado”. Enquanto os padrões anteriores, fazem especificações de endereços dentro da aplicação onde são liberados, como, a URL de autenticação e de criação de conta.

Figura 9 - Exemplo do arquivo de configuração do Spring Security para páginas

```
<security:http use-expressions="true" create-session="always"
    pattern="/**" entry-point-ref="springSocialSecurityEntryPoint"
    xmlns="http://www.springframework.org/schema/security">

    <logout logout-url="/signout" delete-cookies="JSESSIONID" />
    <intercept-url pattern="/favicon.ico" access="permitAll" />
    <intercept-url pattern="/resources/**" access="permitAll" />
    <intercept-url pattern="/signin/**" access="permitAll" />
    <intercept-url pattern="/signup/**" access="permitAll" />
    <intercept-url pattern="/**" access="isAuthenticated()" />

    <security:request-cache ref="httpSessionRequestCache" />


    <access-denied-handler ref="springSocialSecurityAccessDeniedHandler" />

    <anonymous />
    <security:custom-filter position="FORM_LOGIN_FILTER"
        ref="springSocialSecurityAuthenticationFilter" />

    <remember-me services-ref="springSocialSecurityRememberMeServices"
        key="springSocialSecurity" />

</security:http>
```

Fonte: https://github.com/SpringSource/spring-social-samples/tree/master/spring-social-showcase

É possível a definição de múltiplas configurações “http” numa aplicação com Spring Security. Abaixo na figura 10, está um exemplo criado pelo autor para serviços web com arquitetura RESTful, que funciona em conjunto com o exemplo supra.

Figura 10 - Exemplo do arquivo de configuração do Spring Security para Serviços-web

```
<security:http create-session="stateless" pattern="/api/**"
    use-expressions="true">
    <security:intercept-url pattern="/api/admin/**"
        access="hasRole('ROLE_ADMIN')" />
    <security:intercept-url pattern="/api/public/**"
        access="permitAll" />
    <security:intercept-url pattern="/api/signin/**"
        access="isAuthenticated()" />
    <security:intercept-url pattern="/api/tech/**"
        access="hasRole('ROLE_TECH')" />
    <security:intercept-url pattern="/api/usertech/**"
        access="permitAll" />

    <security:http-basic />
</security:http>
```

A diferença para a definição anterior é a utilização da configuração “http-basic” que diz que a autenticação se dá por meio de “Http Basic Authentication”. Dessa forma, todo pedido para endereços protegidos, precisam trazer no cabeçalho HTTP, os dados de autenticação básica. O formato dos dados colocados no cabeçalho são bem simples, apenas o nome do usuário e a senha são concatenados separados por dois pontos e o resultado é codificado para Base64.

2.5.2.3.2 Autenticação pelo próprio GWT

A outra alternativa é fazer a autenticação totalmente na aplicação GWT, sem a utilização de páginas externas. Essa alternativa inteligente foi lida em um blog [21].

A idéia por trás dessa alternativa é a criação de uma Atividade que será responsável por controlar a “View” de autenticação. A “View” possuiria dois componentes para pegar o nome do usuário e a senha, e um botão de autenticação. Quando o botão é acionado, a ação é tratada pelo Apresentador, que poderia ser a

própria Atividade, que irá fazer um pedido HTTP Post e enviar o nome de usuário e a senha para um endereço especificado na configuração do Spring Security.

Caso, a autenticação seja bem sucedida, um status HTTP do tipo OK será retornado, caso contrário, voltará o status HTTP do tipo BAD CREDENTIALS.

Abaixo está a configuração do Spring Security. O ponto de entrada do Spring Security seria inexistente e, por isso, apontaria para um filtro que sempre retorna o HTTP NOT AUTHORIZED. A presença de todos esses filtros é necessária, já que a aplicação GWT não funciona por meio de páginas, então a única forma de comunicar o resultado da operação é através dos códigos HTTP.

Figura 11 - Exemplo do arquivo de configuração do Spring Security

```
  <http auto-config="true" entry-point-ref="http401UnauthorizedEntryPoint" create-session="always">
      <form-login authentication-success-handler-ref="authenticationSuccessHandler"
                  authentication-failure-handler-ref="authenticationFailureHandler"/>
      <logout success-handler-ref="logoutSuccessHandler"/>

      <custom-filter before="CONCURRENT_SESSION_FILTER" ref="XSRFAttackFilter" />
</http>

<beans:bean id="XSRFAttackFilter" class="com.myappenginecookbook.security.XSRFAttackFilter"/>

  <!-- Use this entry point to signal to the GWT-caller that the user needs to log in to access the resource-->
  <beans:bean id="http401UnauthorizedEntryPoint"
      class="com.myappenginecookbook.security.Http401UnauthorizedEntryPoint" />


  <beans:bean id="authenticationSuccessHandler" class="com.myappenginecookbook.security.GWTAuthenticationSuccessHandler"/>
  <beans:bean id="authenticationFailureHandler" class="com.myappenginecookbook.security.GWTAuthenticationFailureHandler"/>
  <beans:bean id="logoutSuccessHandler" class="com.myappenginecookbook.security.GWTLogoutSuccessHandler"/>

    <authentication-manager>
      <authentication-provider>
        <user-service>
          <user name="jimi" password="jimispassword" authorities="ROLE_USER, ROLE_ADMIN" />
          <user name="bob" password="bobspassword" authorities="ROLE_USER" />
        </user-service>
      </authentication-provider>
    </authentication-manager>

  <!-- Secure all beans loaded into the ApplicationContext -->
  <global-method-security
      secured-annotations="enabled" jsr250-annotations="disabled" />

  <!-- Automatically receives AuthenticationEvent messages -->
  <beans:bean id="loggerListener" class="org.springframework.security.authentication.event.LoggerListener"/>
```

Fonte: http://technowobble.blogspot.com.br/2010/05/gwt-and-spring-security.html

2.5.3 Segurança por Código

Independente da alternativa escolhida, como já detectado num capítulo anterior, para saber se o usuário está autenticado, a própria aplicação terá de

perguntar para a camada de serviço, que é confiável. Para fazer isso, a melhor alternativa é aproveitar a arquitetura do GWT que faz o uso de uma Atividade nova toda vez que um usuário entra em uma URL.

Para isso, será necessário a criação de uma Atividade base para todas as outras Atividades, que irá perguntar à camada de serviço se o usuário está ou não autenticado. Um exemplo disso pode ser observado abaixo.

A atividade “Base Activity” será responsável por perguntar para o serviço se o usuário está ou não autenticado. Sendo positiva a resposta, manda, através do “EventBus”, um evento informando que o usuário está autenticado. Se negativo, manda um evento informando o contrário. Esses eventos podem ser úteis, pois podem existir Atividades funcionando em paralelo que queiram saber se o usuário está autenticado ou não, para mostrar, por exemplo, dados personalizados. Também, na hipótese de usuário que não esteja autenticado, o “PlaceController” é acionado para mandar o usuário para um local para autenticação.

Caso todas as atividades da aplicação sejam superclasses dessa “BaseActivity”, essa proteção de lugares que cada usuário pode acessar será feita. Não será muito eficiente, já que a cada nova atividade, um novo pedido HTTP deve ser feito de forma assíncrona, mas é algo necessário. Supondo que esse sistema não fosse utilizado, haveria o risco de um usuário desativado estar utilizando o sistema e tendo acesso a lugares que não deveria.

Como já discutido anteriormente, a solução anterior é para a questão gráfica da segurança, ou seja, questões de acesso a lugares diferentes da aplicação. Mas como já mostrado também, isso não será o suficiente, já que o código em Javascript poderia ser modificado por um atacador, fazendo a chamada assíncrona não ser realizada, por exemplo.

Para resolver esse problema, o Spring Security oferece uma boa solução: anotações. Na camada de serviço da aplicação pode-se definir através de simples anotações, restrições de acesso a métodos. Isso vai possibilitar um controle eficiente dos dados.

Através da anotação disponibilizada pelo Spring “@PreAuthorize”, pode-se especificar qual a restrição necessária para acessar aquele método. Então, se alguém possuir o “Role”, um papel, “ROLE_USER”, que caracteriza um usuário comum do sistema, terá acesso, caso contrário, será enviado uma resposta de HTTP BAD CREDENTIALS.

2.5.4 Persistência dos Usuários

Além da configuração do que é restrito e o que é aberto na aplicação, ainda existe outra questão importante de como serão armazenados os dados dos usuários. Para isso, precisa-se entender um pouco de como o Spring Security funciona.

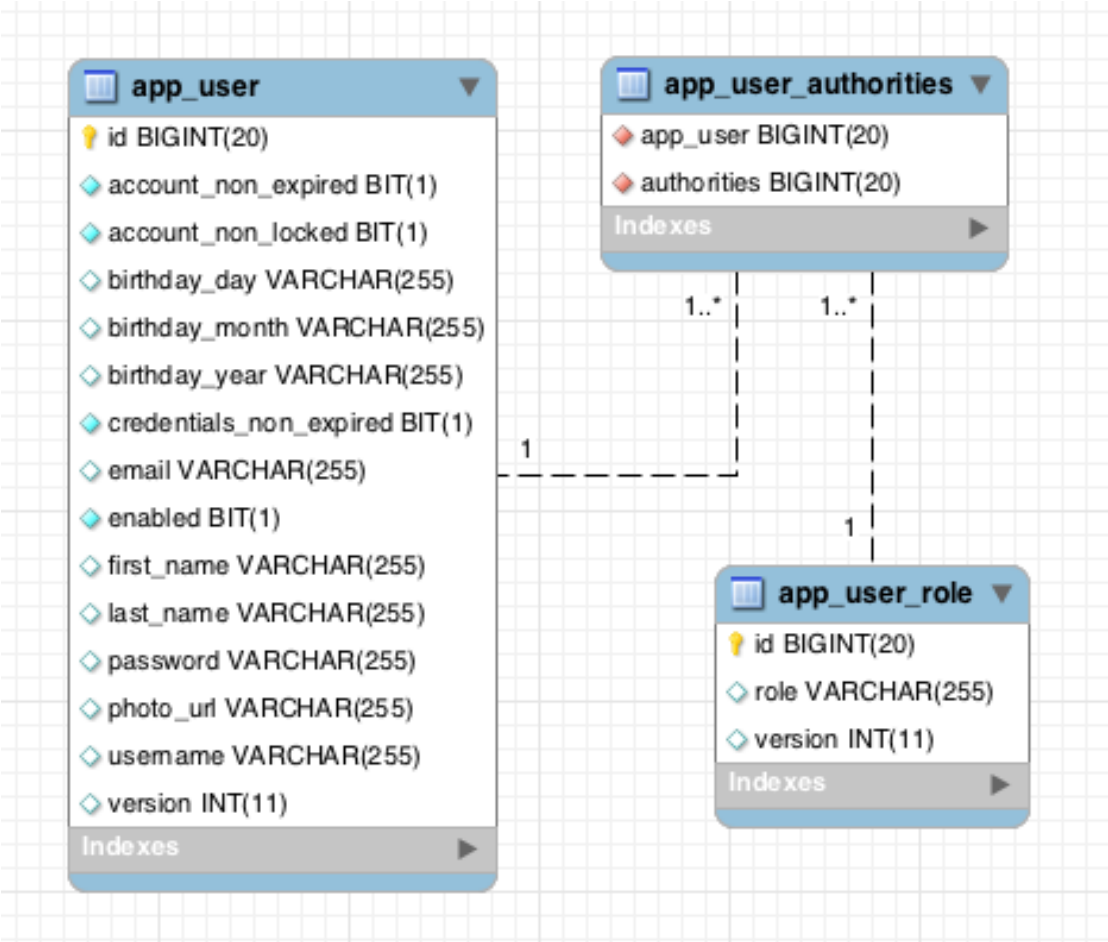
O próprio Spring Security é responsável por gerenciar os detalhes de autenticação de um usuário na sessão. O que se precisa saber é que isso é feito através de um objeto que implementa a interface “Authentication”. Para a criação de um objeto que implemente essa interface Java, há a restrição de que exista um nome de usuário, e uma chave, possivelmente a senha do usuário, e uma lista das autoridades do usuário.

Isso tudo irá provocar a necessidade de salvar, no mínimo, essas três informações em uma estrutura de dados. Por meio da interface “UserDetails”, são definidos os dados que são esperados de um usuário do sistema, que serão encapsulados no objeto “Authentication” na sessão. Uma solução para o

armazenamento desses dados é através da criação de um objeto que implemente a “UserDetails”, representando um usuário. Além desse objeto, precisa um outro que implemente a interface “GrantedAuthority” para representar as autoridades, ou os papéis, dos usuários no sistema.

Abaixo na figura 12, está definido um modelo de como isso seria armazenado em um banco de dados. Através da entidade “AppUser”, define-se um usuário, enquanto a entidade “AppUserRole” representa um papel do usuário no sistema. Um usuário poderá possuir diversos papéis no sistema e no mínimo um papel, sendo que isso está representado pelo relacionamento “Muitos-para-Muitos” na tabela “app_user_authorities”.

Figura 12 - Exemplo de modelo relacional para representar usuários e seus papéis num sistema



Para fazer o Spring Security entender de onde ele deve obter esses dados, deve-se implementar a interface “UserDetailsService”, que possui apenas um método para obter um usuário através de seu nome de usuário. Define-se então que essa classe que implementa a interface, será parte do provedor de autenticação, através de uma linha na configuração do “ApplicationContext-security” como abaixo:

<security:authentication-provider user-service-ref="customerUserDetailsService"/>

De forma diferente, pode-se implementar o próprio provedor de autenticação, através da interface Java “AuthenticationProvider”, e a utilização da seguinte definição abaixo na figura 13.

Figura 13 - Exemplo do arquivo de configuração do Spring Security

```
<security:authentication-manager alias="authenticationManager">
    <security:authentication-provider
        ref="customAuthenticationProvider" />
</security:authentication-manager>
```

A implementação é trivial, existe apenas um método chamado “authenticate”.

2.5.5 Conclusões sobre Segurança com GWT

Através do Spring Security, depois de muita pesquisa, conseguiu-se uma maneira de utilizar esse framework para proteger uma aplicação GWT. A principal vantagem da utilização desse é a integração com o Spring Framework que pode ser responsável pela camada de serviço da aplicação e existe a fácil possibilidade de

estender a segurança em um novo cliente da aplicação, como, por exemplo, através do uso de um serviço web protegido.

O maior ponto negativo é a falta de documentação disponível em relação a estratégias de como configurar o Spring Security para funcionar com uma aplicação GWT, ou seja, não foi encontrado um passo a passo, mas diversas ideias interessantes.

2.6 Conclusão

Durante esse capítulo, estudos foram feitos, através de leitura, desenvolvimento e análise de código, relacionados a como criar uma aplicação web utilizando tecnologias da Spring Source para Java e o GWT. Os seguintes detalhes puderam ser observados:

- O Spring Roo, como já concluído anteriormente, é a tecnologia para o ponto de partida do projeto, sendo desligada em seguida para que o desenvolvimento possa ser acelerado.
- A arquitetura SOA, como já esperado, é extremamente eficiente, pois irá separar a parte mais importante da aplicação, ou seja, a camada responsável pela lógica de negócio e que controla a entrada e saída de dados, do restante da aplicação. Clientes em Android podem ser desenvolvidos para a aplicação, através do uso de serviços web REST, enquanto um cliente em GWT também pode ser desenvolvido através do uso do mecanismo “Request Factory” do GWT.
- O Spring Security é um framework extremamente poderoso para autenticação, permitindo simples anotações para proteger a camada de serviço e isso já elimina uma quantidade grande de trabalho quanto a questão da autorização.

Mesmo não sendo encontrado um passo a passo para configurar esse framework no GWT, boas ideias foram encontradas e puderam ser aplicadas com sucesso.

- Já o cliente desenvolvido em GWT é um cliente extremamente pesado em termos de desenvolvimento. Muitas linhas de código são necessárias para pouca produção, caso o framework “Places and Activities” da versão 2.4 do GWT seja utilizado.

3 PROJETO LÓGICO

Esse capítulo abordará os materiais e métodos que serão utilizados no desenvolvimento do projeto, bem como será proposto o projeto lógico do sistema a ser desenvolvido, ou seja, uma visão geral do sistema, requisitos e casos de uso de cada componente do sistema e a modelagem de dados do sistema.

A primeira questão a ser tratada é em relação a como os dados serão representados. Já que o sistema a ser desenvolvido deve ser compatível com o modelo de dados do software Google Transit, existe a necessidade de entender o modelo de dados GTFS-realtime.

Em seguida, serão definidos os requisitos e casos de uso do sistema a ser desenvolvido e o projeto proposto por esse trabalho. Por fim, os métodos e materiais que serão utilizados no projeto proposto serão apresentados.

3.1 Modelo de dados

3.1.1 Representação dos dados de transporte público

Baseado em JORGE (2011) e no estudo feito a respeito dos modelos de representação de dados de trânsito, será utilizado neste trabalho a especificação GTFS, com a diferença dessa apresentar a extensão para dados em tempo real.

GTFS ou “Google Transit Data Feed Specification” é uma especificação para representação de dados de agências de transporte público. Essa especificação permite que uma agência informe ao público a respeito do calendário de viagens programadas. Mais detalhes podem ser observados em JORGE (2011) ou KIZOOM (2008) e no capítulo seguinte.

Já o GTFS-realtime é uma extensão do GTFS e é uma especificação que permite que agências de transporte publiquem os dados em tempo real de suas frotas.

Em JORGE (2011), a aplicação apresentada utilizou a especificação GTFS, ou seja, tinha como objetivo a manipulação de dados estáticos a respeito do transporte público. Já, através do GTFS-realtime, o modelo de dados representados deve ser ampliado para suportar dados como a posição e a velocidade de um veículo, o nível de congestionamento, alertas e outros dados que serão descritos em seguida.

O GTFS-realtime suporta três tipos de mensagens: “TripUpdate”, “VehiclePosition” e “Alert”. “TripUpdate” dá o poder a agência de transporte público notificar o público a respeito de atrasos em viagens, ou se uma viagem está seguindo sua programação, ou o veículo responsável pela viagem, enquanto o “VehiclePosition” permite que a posição e a velocidade de um veículo seja divulgada. Por fim, a “Alert” irá permitir que avisos internacionalizados sejam publicados quando imprevistos acontecerem. A especificação detalhada dessas novas mensagens pode ser observada no ANEXO A.

3.1.2 Modelo de Dados Proposto

Em KIZOOM (2008), pode-se observar um modelo entidade relacionamento para representação do GTFS. Até o momento, não foram encontradas propostas de um modelo para o GTFS-realtime. Diante disso, a seguir, será descrito o modelo compatível com GTFS-realtime que será utilizado no sistema a ser proposto, considerando as três novas mensagens adicionadas ao GTFS na especificação

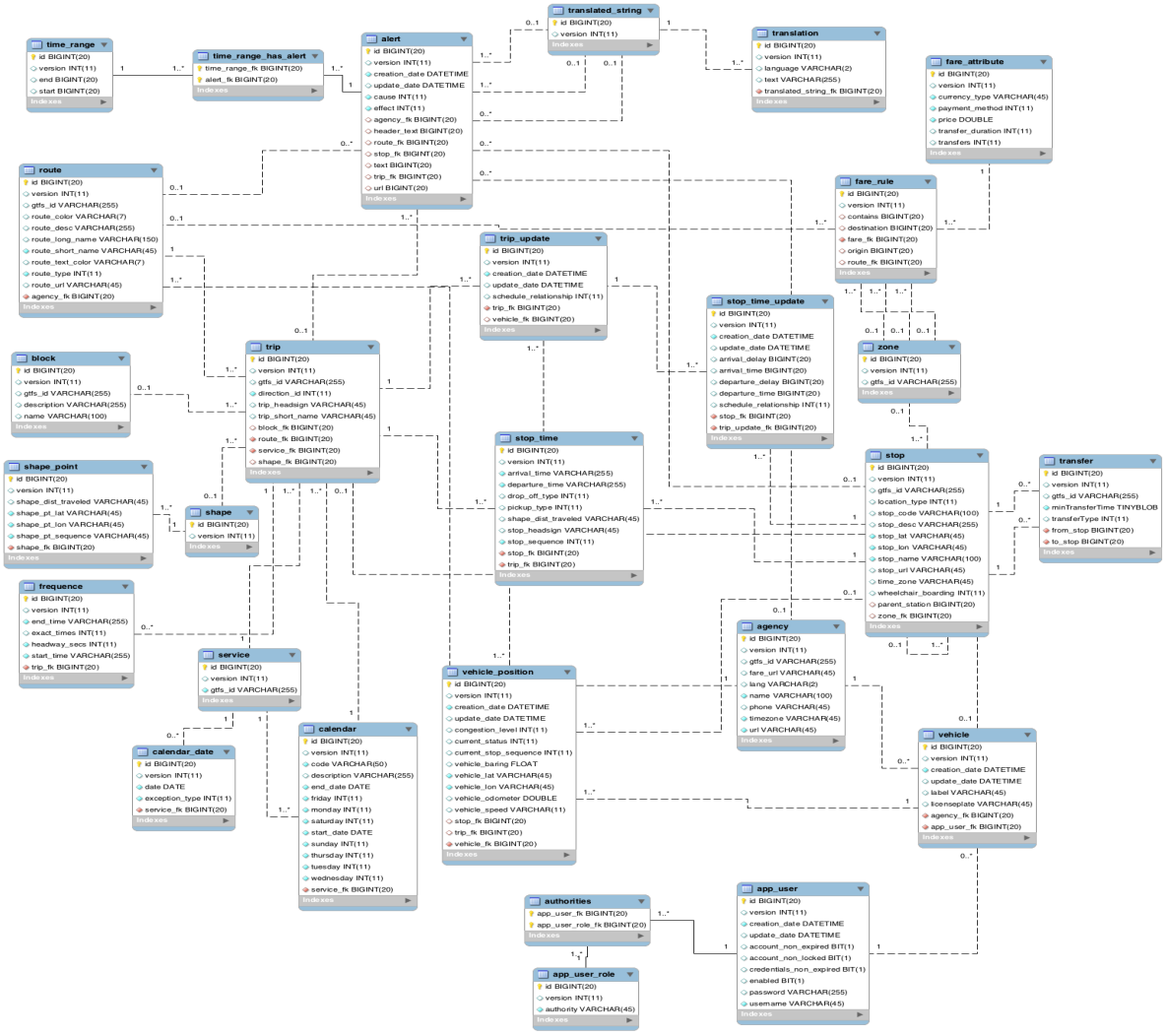
GTFS-realtime e a utilização do modelo encontrado em KIZOOM (2008, p.19) como base.

Para poder entender o modelo proposto, é necessário observar os seguintes conceitos em relação ao modelo GTFS: “agency” representa uma agência de transporte público, “stop” representa uma parada de um veículo, “route”, ou rota, representa um grupo de “trips”, “trips”, ou viagem, representa uma sequencia de duas ou mais paradas de veículo, “stop_times” representa o momento em que um veículo irá parar em uma viagem, e “calendar” representa o tempo de funcionamento de um serviço. Outros conceitos presentes no GTFS como “transfer”, “zone, entre outros serão ignorados por não serem obrigatórios nessa representação de dados.

O modelo proposto é simplesmente uma extensão daquele apresentado em KIZOOM (2008, p.19). Devido à adição de três novos tipos de mensagem no GTFS-realtime, novas entidades tiveram de ser criadas para representação dessas. Somado a isso, também foi necessário modelar os veículos.

Abaixo, na figura 14, está o modelo relacional, sendo que as novas adições serão analisadas, individualmente, em seguida através de grafos de relacionamento gerados pelo programa “SchemaSpy”[25].

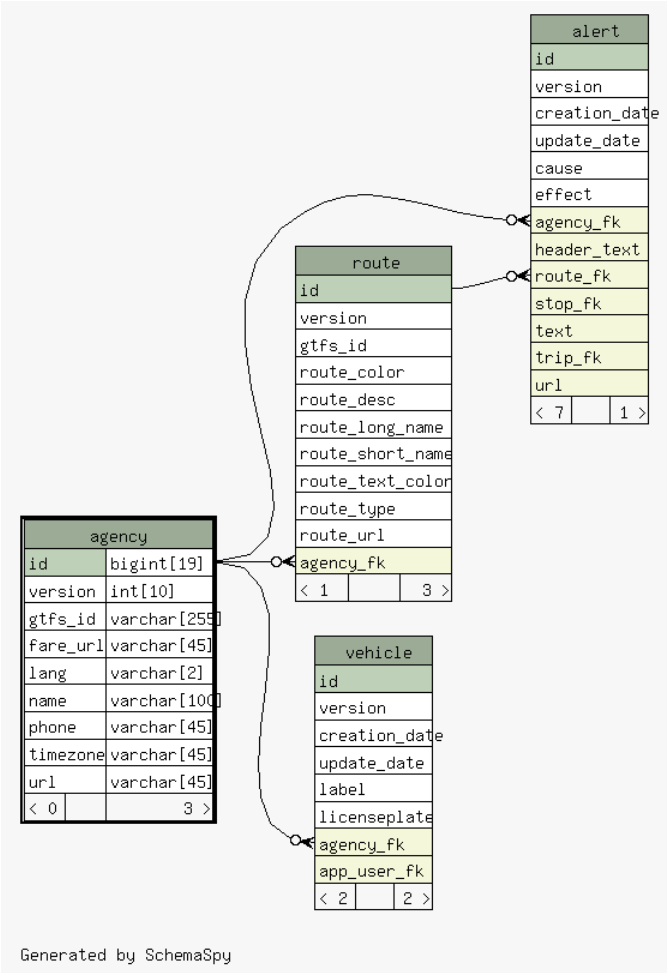
Figura 14 - Modelo Relacional GTFS-realtime



Modelo Relacional GTFS-realtime

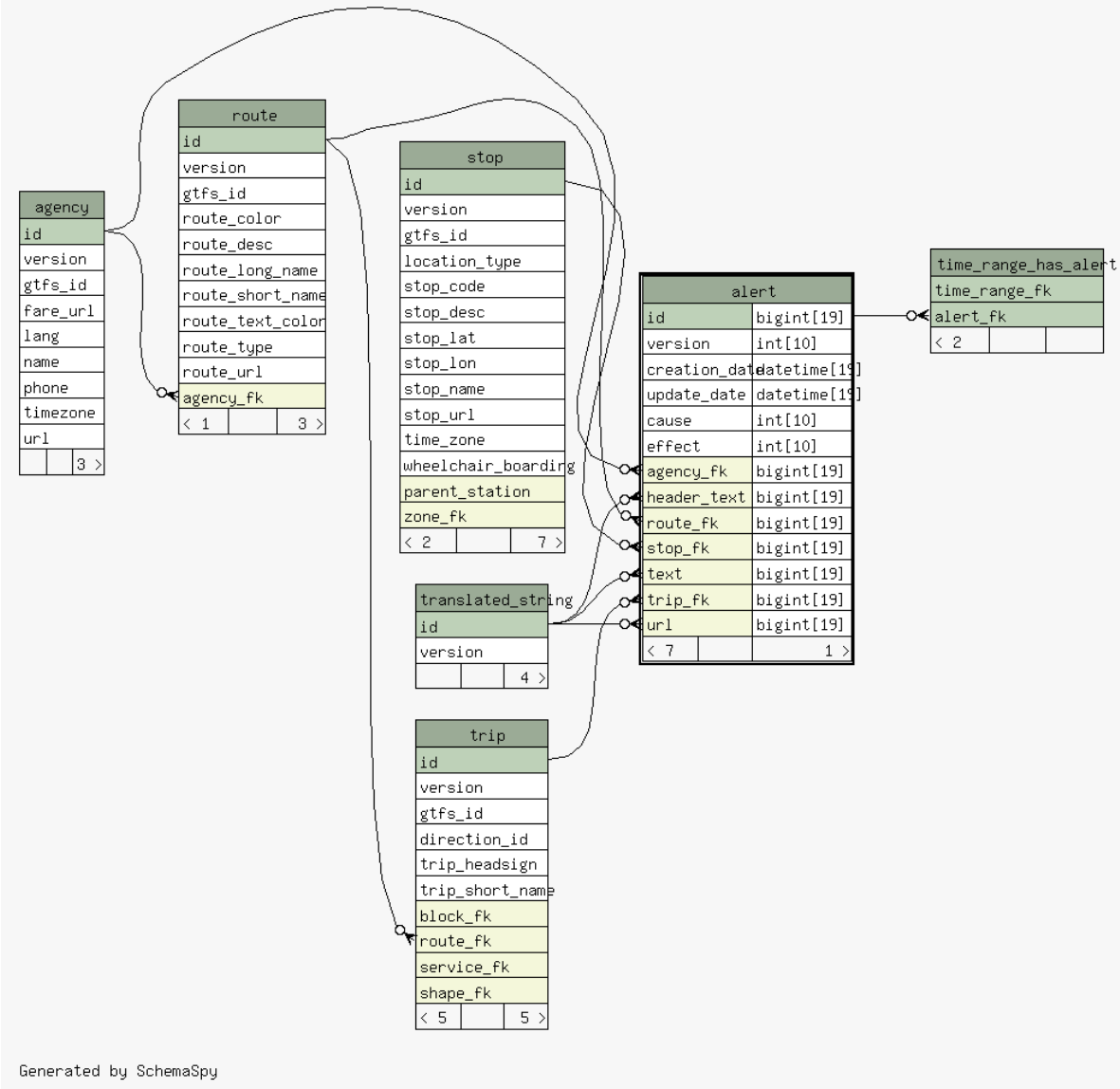
Na figura 15, pode-se observar o grafo da tabela “agency”, que representa uma agência de trânsito, e suas relações diretas com outras tabelas no modelo. Uma agência poderá ter diversas rotas, representadas pela tabela “route”, diversos alertas, representados pela tabela “alert”, e diversos veículos, representados pela tabela “vehicle”.

Figura 15 - Grafo de relacionamentos da tabela Agency



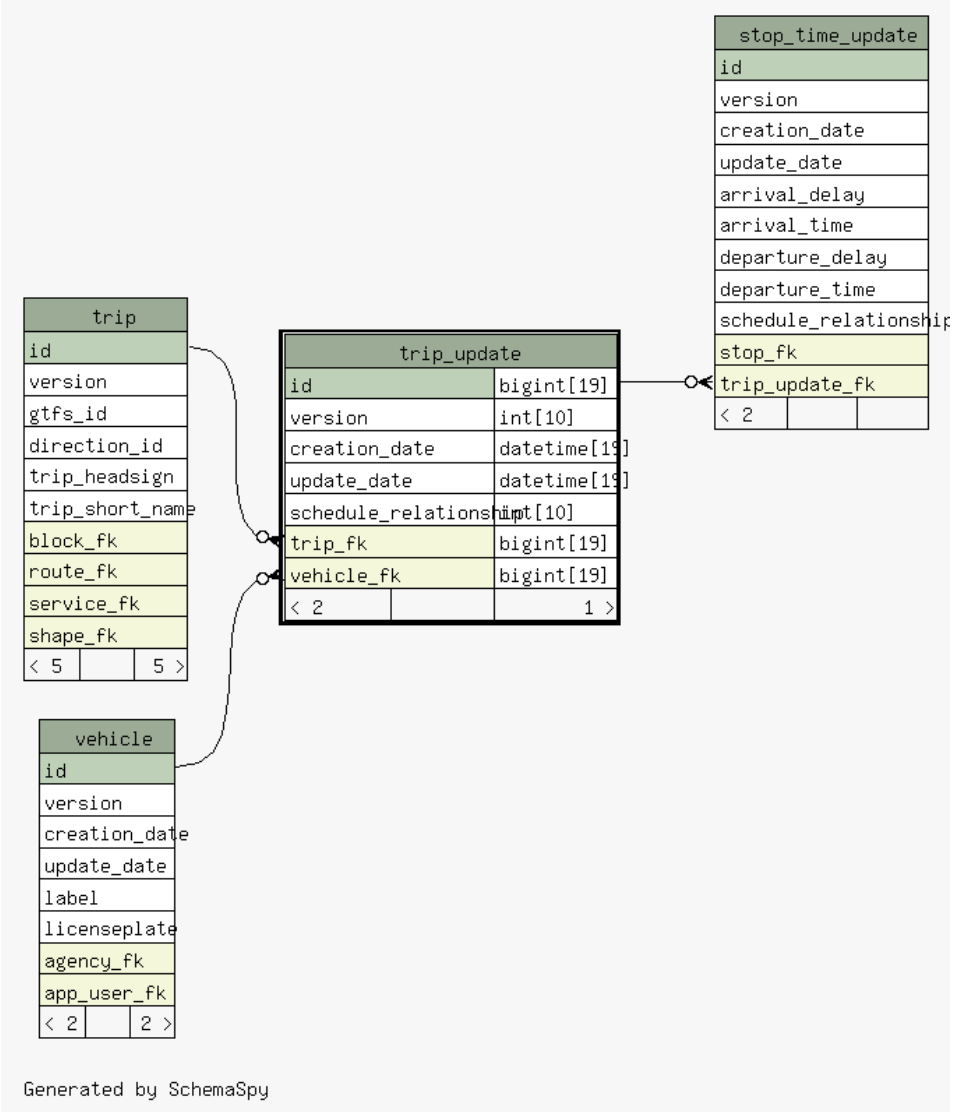
Já na figura 16, pode-se observar o grafo da tabela “alert”, que representa os alertas, especificados no GTFS-realtime, que podem ser emitidos para diversos escopos: um alerta pode ser válido para uma agência, ou uma rota, ou uma parada de ônibus, representada pela tabela “stop”, ou uma viagem, representada pela tabela “trip”. A tabela “alert” não foi normalizada devido a questões de performance: caso se conheça uma viagem, por exemplo, seria mais fácil fazer uma busca de quais alertas são válidos para essa, já que tudo poderia ser resolvido diretamente na cláusula “where” indicando a agência, a rota e a própria viagem como alternativas, sem a necessidade de junções.

Figura 16 - Grafo de relacionamentos da tabela Alert



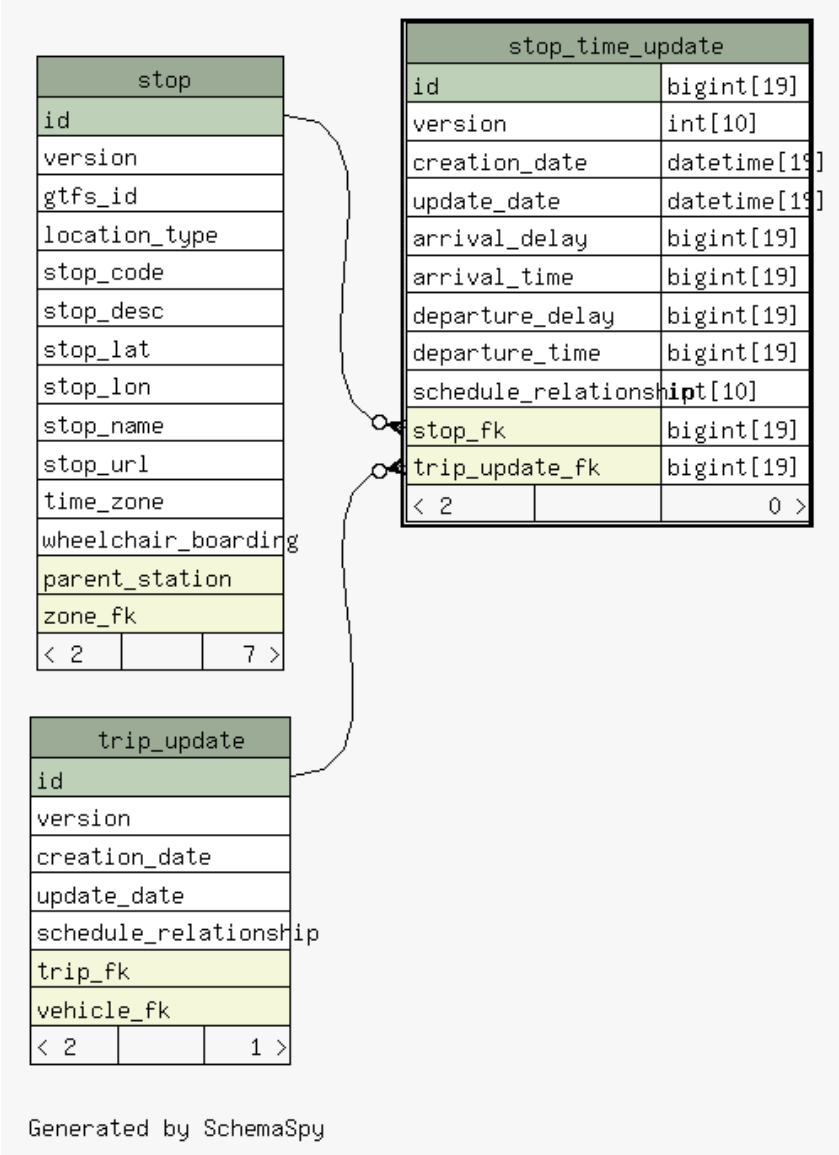
Na figura 17, pode-se observar o grafo da tabela “trip_update”, que representa a mensagem de atualização de uma viagem especificada no GTFS-realtime. Essa atualização é utilizada quando agências querem informar ao público a respeito de atrasos de viagens, ou o veículo que está efetuando a viagem, ou caso a viagem tenha sido cancelada ou adicionada. Essa tabela deve possuir uma relação com uma viagem, pode possuir uma relação com um veículo, e pode possuir diversas relações com a tabela “stop_time_update”.

Figura 17 - Grafo de relacionamentos da tabela Trip Update



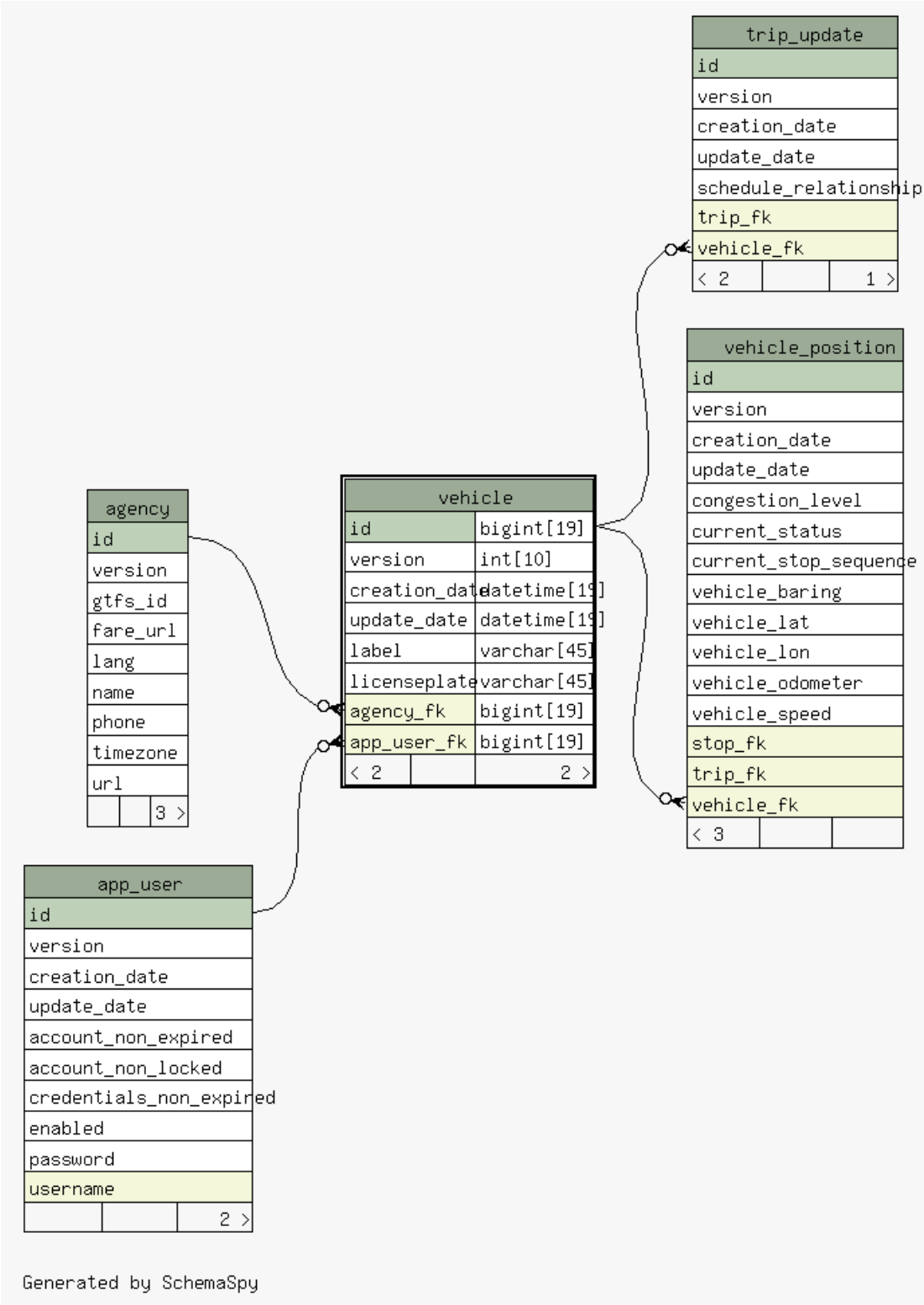
Na figura 18, pode-se observar o grafo da tabela “stop_time_update” e suas relações diretas. Essa tabela representa uma atualização em relação a uma única parada durante uma viagem. Ou seja, significa o estado atual de uma parada de ônibus em relação a viagem. A parada nessa estação de ônibus pode ser cancelada, ou adicionada ou sofrer atrasos. Essa tabela representará parte de informação da mensagem de atualização de uma viagem especificada pelo GTFS-realtime. Possuirá relação com a tabela “trip_update” e com a tabela “stop”, que representa uma parada de ônibus.

Figura 18 - Grafo de relacionamentos da tabela Stop Time Update



A representação de um veículo é exibida na figura 19. Um veículo deve possuir uma agência, e um usuário no sistema, representado pela tabela “app_user”. Poderá possuir diversas atualizações de viagem, tabela “trip_update” e atualizações de sua localização em um determinado instante, tabela “vehicle_position”.

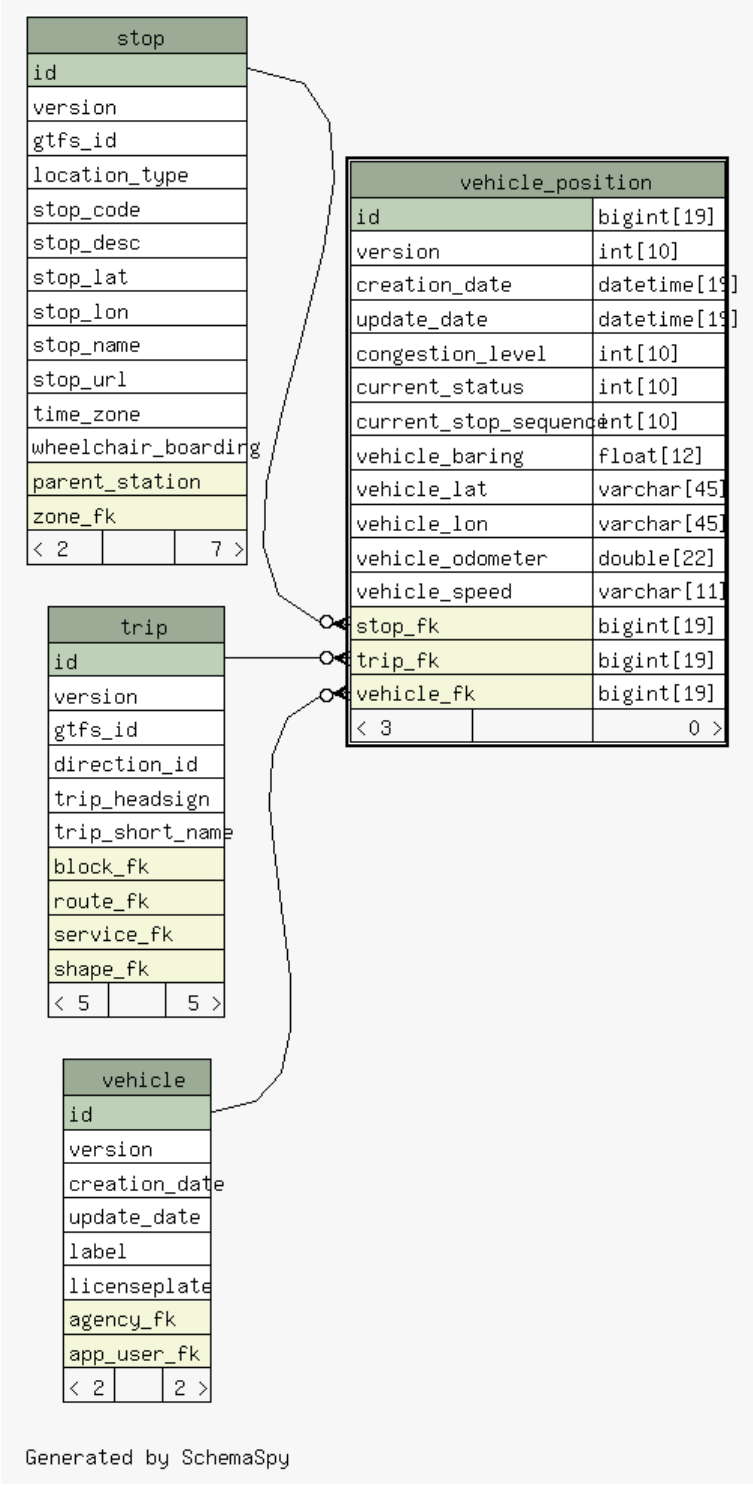
Figura 19 - Grafo de relacionamentos da tabela Vehicle



Na figura 20, por fim, podemos observar a representação relacional da última mensagem especificada no GTFS-realtime: “VehiclePosition”. Essa mensagem, representada pela tabela “vehicle_position”, informa o público a respeito

da posição de um veículo em um determinado instante. Portanto, a tabela possui relação com a tabela “vehicle”.

Figura 20 - Grafo de relacionamentos da tabela Vehicle Position

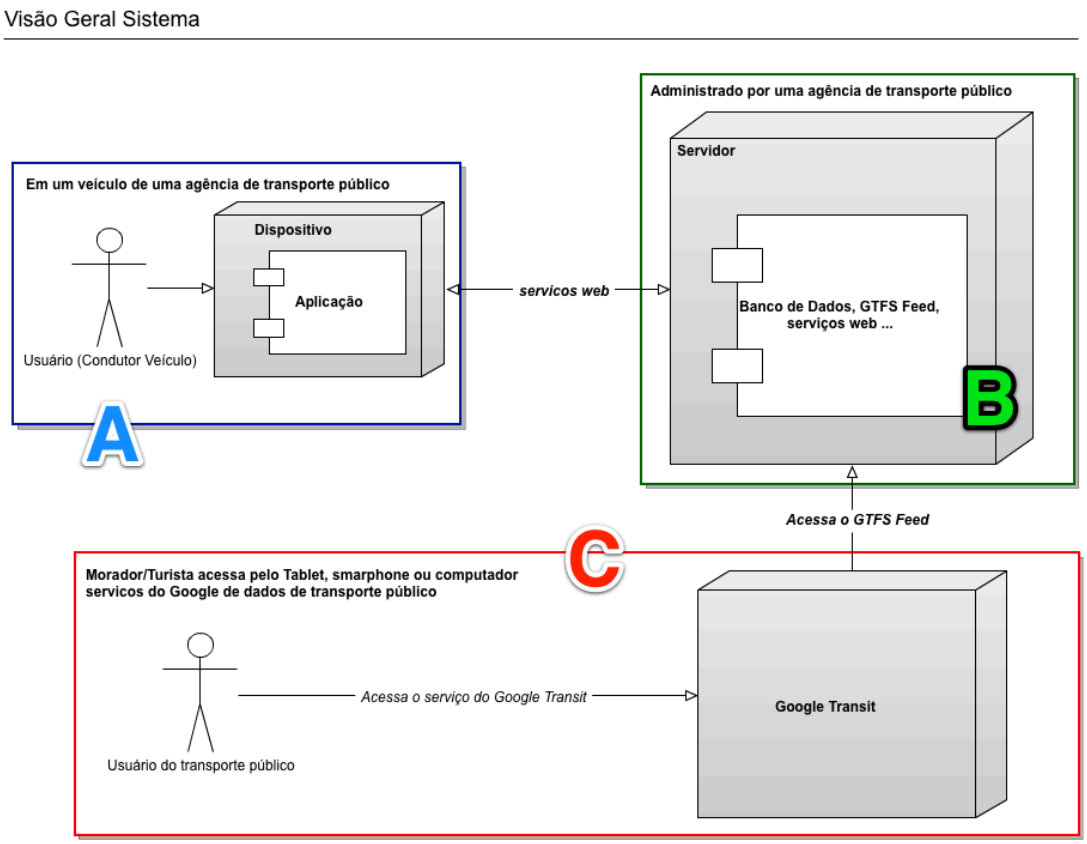


3.2 Sistema Proposto

O sistema proposto pode ser dividido em dois grandes componentes: primeiramente um servidor que é responsável por armazenar todos os dados em tempo real de veículos no formato GTFS-realtime e disponibilizar para o serviço Google Transit, e por fim, o cliente do servidor que estará sendo executado em um dispositivo dentro de um veículo e terá função de transmitir os dados em tempo real para o servidor. Os dados que serão enviados ao servidor pela aplicação cliente são todos aqueles requeridos pelas mensagens do GTFS-realtime, mas, facilmente, pode-se ampliar de acordo com as necessidades futuras.

A figura 21 abaixo representa uma visão geral do sistema. O componente descrito em azul com a letra “A” representa a aplicação cliente do servidor web, que está representado em verde pela letra “B”. Já em “C”, em vermelho, observa-se como o sistema externo do Google Transit irá interagir com o servidor web.

Figura 21 - Visão Geral do sistema proposto



A aplicação cliente em “A” estará rodando em um dispositivo dentro de um veículo de uma agência responsável por transporte público em uma cidade. O condutor do veículo, ou, alguma outra pessoa, seria responsável por informar, através da aplicação, alertas a respeito de problemas encontrados durante o serviço do transporte como, por exemplo, acidentes. A aplicação cliente fica responsável de comunicar a geo-posição do veículo. Informações geradas por essa serão transmitidos para os serviços web do servidor em “B” através da internet.

A aplicação servidor em “B” será responsável por armazenar os dados originados da aplicação cliente que está sendo executada em um veículo de uma agência de transporte público. O servidor poderá representar dados de uma ou mais agências. O servidor irá possuir uma base de dados relacional, irá oferecer seus

dados no formato do Google Transit através de serviços web, assim como para receber dados da aplicação cliente.

Em “C”, observa-se que o sistema do Google irá consultar o servidor em “B” para obter dados da agência. Isso irá possibilitar que usuários do sistema de transporte público da cidade possam acessar aplicativos do Google para buscar informações a respeito dos serviços. Uma alternativa a essa representação em “C”, seria o desenvolvimento por uma agência de uma aplicação que irá, através dos dados obtidos do servidor, mostrar todas essas informações para os usuários dos serviços públicos.

3.2.1 Características Gerais dos Componentes

Como já mencionado, os dois componentes que serão desenvolvidos são um cliente que será executado em um dispositivo móvel dentro de um veículo e um servidor que irá receber os dados do cliente e disponibilizará para os serviços do Google. Em seguida serão citadas as funções gerais de cada componente.

O servidor que, será chamado a partir desse capítulo de SERVIDOR, será responsável pelas seguintes funções:

- Armazenar dados das agências de trânsito e seus veículos;
- Disponibilizar dados para o Google Transit no formato GTFS-realtime;
- Receber dados relacionados as mensagens do GTFS-realtime de aplicações clientes instaladas em veículos;
- Precisará de um subcomponente chamado de CLIENTE WEB para permitir o cadastro de agências, rotas, veículos, e outros dados requeridos para modelar a estrutura de dados de um sistema de transporte público no formato GTFS;

Já o cliente no veículo, que será chamado a partir desse capítulo de CLIENTE MÓVEL, terá a seguinte função:

- Transmitir dados requeridos pelo GTFS-realtime como a geo-localização de um veículo e alertas.

3.2.2 Requisitos dos Componentes

3.2.2.1 Requisitos do SERVIDOR

Abaixo estão definidos os requisitos funcionais do SERVIDOR:

REF1. Receber a geo-localização de um veículo.

REF2. Receber alertas de um veículo.

REF3. Disponibilizar arquivos no formato GTFS e GTFS-realtime acessíveis através de serviços web para todo e qualquer usuário na internet.

REF4. Disponibilizar as operações de cadastrar e listar entidades que representam agências, viagens, veículos, rotas, paradas de veículos e disponibilidade de serviços através de serviços web protegidos, ou seja, restritos a usuários conhecidos pelo sistema.

Abaixo estão definidos os requisitos não-funcionais do servidor:

RNF1. Servidor deverá ser feito em Java, utilizando o framework Spring Framework.

RNF2. Servidor deverá utilizar o MySQL como tecnologia do banco de dados relacional.

RNF3. Servidor deve utilizar REST como tecnologia dos serviços web para comunicação com o CLIENTE MÓVEL.

3.2.2.2 Requisitos do CLIENTE WEB

REF1. O sistema deve permitir a criação e a listagem de agências.

REF2. O sistema deve permitir a criação e a listagem de veículos.

REF3. O sistema deve permitir a criação e a listagem de viagens.

REF4. O sistema deve permitir a criação e a listagem de rotas.

REF5. O sistema deve permitir a criação e a listagem de paradas de veículos.

REF6. O sistema deve permitir a criação e a listagem de disponibilidades de serviço.

REF7. O sistema deve permitir a criação de usuários.

REF8. O sistema deve permitir a autenticação de um usuário cadastrado no mesmo.

RNF1. O sistema deve ser feito com a tecnologia GWT.

3.2.2.3 Requisitos da Aplicação CLIENTE MÓVEL

Abaixo estão definidos os requisitos funcionais da aplicação CLIENTE MÓVEL:

REF3. O sistema deve permitir a autenticação de um usuário do tipo veículo cadastrado no sistema CLIENTE SERVIDOR.

REF2. O sistema deve enviar a geo-localização do veículo a cada 10 segundos.

REF3. O sistema deve permitir que o usuário escreva e envie alertas e escolha o tipo de alerta e o efeito desse conforme definido pelo GTFS-realtime na mensagem “Alert”.

Abaixo estão definidos os requisitos não-funcionais da aplicação cliente móvel:

RNF1. O sistema deve ser feito para o sistema operacional Android.

3.2.3 Casos de Uso

Os Casos de Uso de cada aplicação do projeto proposto serão apresentados seguindo a estrutura casual descrita por Allistar Cockburn [29]: um Caso de Uso deve possuir um título, ator primário, escopo, nível e uma descrição informal. Somado a isso, serão atribuídos códigos a cada Caso de Uso para facilitar a identificação do mesmo.

Os escopos utilizados nos Casos de Uso serão os seguintes: VEÍCULO, SERVIÇO e ADMINISTRADOR, VISUALIZADOR DE DADOS. O escopo VEÍCULO descreve aqueles Casos de Uso responsáveis por algo que envolve usuários com esse papel. O escopo SERVIÇO descreve Casos de Uso que representam um ponto de entrada para a camada de serviço, normalmente operações CRUD [30]. Já o escopo VISUALIZADOR DE DADOS representa operações relacionadas aos dados no formato GTFS e GTFS-realtime. Por fim, o escopo ADMINISTRADOR é utilizado para descrever tarefas que somente um usuário responsável por uma Agência poderia fazer, ou seja, criação de dados associados ao transporte público.

Esses ESCOPOS poderão ser diretamente mapeados para papés de usuários no sistema, por exemplo.

3.2.3.1 Casos de Uso da Aplicação SERVIDOR

Abaixo seguem os Casos de Uso da aplicação SERVIDOR.

Caso de Uso 1.1 – Publicar Alerta (CLIENTE MÓVEL / escopo veículo)

O CLIENTE MÓVEL envia, através do HTTP, um arquivo no formato “application/json” para o sistema descrevendo um alerta. O sistema responde HTTP OK caso os dados estejam válidos e o processo seja um sucesso. Caso contrário, responde HTTP BAD REQUEST.

Caso de Uso 1.2 – Publicar Posição do Veículo (CLIENTE MÓVEL / escopo veículo)

O CLIENTE MÓVEL envia, através do HTTP POST, um arquivo no formato “application/json” para o sistema descrevendo a posição atual do veículo. O sistema responde HTTP OK caso os dados estejam válidos e o processo seja um sucesso. Caso contrário, responde HTTP BAD REQUEST.

Caso de Uso 1.3 – Obter Feed VehiclePosition (GOOGLE / escopo visualizador de dados)

O GOOGLE faz um pedido HTTP GET para uma URL em que indica um identificador de uma Agência. O sistema responde HTTP OK e um arquivo no formato “application/x-google-protobuf” contendo um feed com as mensagens VehiclePosition caso a agência exista. Caso contrário, responde HTTP NOT FOUND.

Caso de Uso 1.4 – Obter Feed Alert (GOOGLE / escopo visualizador de dados)

O GOOGLE faz um pedido HTTP GET para uma URL em que indica um identificador de uma Agência. O sistema responde HTTP OK e um arquivo no

formato “application/x-google-protobuf” contendo um feed com as mensagens Alert, caso a agência exista. Caso contrário, responde HTTP NOT FOUND.

Caso de Uso 1.5 – Obter Feed TripUpdate (GOOGLE / escopo visualizador de dados)

O GOOGLE faz um pedido HTTP GET para uma URL em que indica um identificador de uma Agência. O sistema responde HTTP OK e um arquivo no formato “application/x-google-protobuf” contendo um feed GTFS-realtime com mensagens do tipo TripUpdate, caso a agência exista. Caso contrário, responde HTTP NOT FOUND.

Caso de Uso 1.6 – Obter GTFS.zip (GOOGLE / escopo visualizador de dados)

O GOOGLE faz um pedido HTTP GET para uma URL em que indica um identificador de uma Agência. O sistema responde HTTP OK e um arquivo no formato “application/zip” contendo os dados especificados pelo GTFS, caso a agência exista. Caso contrário, responde HTTP NOT FOUND.

Caso de Uso 1.7 – Salvar Veículo (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de um Veículo e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.8 – Pegar Veículos (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o

HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todos Veículos encontrados. Caso aconteça um erro, HTTP INTERNAL ERROR OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.9 – Salvar Rota (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de uma Rota e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.10 – Pegar Rotas (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todas Rotas encontradas. Caso aconteça um erro, HTTP INTERNAL ERROR OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.11 – Salvar Disponibilidade de Serviço (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de uma Disponibilidade de Serviço e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário

fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.12 – Pegar Disponibilidades de Serviço (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todas Disponibilidades de Serviço encontradas. Caso aconteça um erro, HTTP INTERNAL ERROR OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.13 – Salvar Parada de Veículo (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de uma Parada de Veículo e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.14 – Pegar Paradas de Veículos (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o

HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todas Paradas de Veículo encontradas. Caso aconteça um erro, HTTP INTERNAL ERROR OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.15 – Salvar Viagem (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de uma Viagem e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.16 – Pegar Viagens (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todas Viagens encontradas. Caso aconteça um erro, HTTP INTERNAL ERROR OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.17 – Salvar Usuário (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de um Usuário e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam

salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.18 – Pegar Usuários (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todos Usuários encontradas. Caso aconteça um erro, HTTP INTERNAL ERROR OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.19 – Salvar Agência (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP POST para uma URL, contendo um arquivo “application/json” que descreve os dados de uma Agência e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK caso os dados sejam válidos e os dados sejam salvos corretamente. Caso contrário, responde HTTP BAD REQUEST se algum dado é inválido, ou HTTP INTERNAL ERROR se um erro acontecer e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.20 – Pegar Agências (CLIENTE WEB / escopo serviços)

O CLIENTE WEB faz um pedido HTTP GET para uma URL, contendo opcionalmente parâmetros de busca para limitar o número de resultados e o HEADER AUTHORIZATION do HTTP indicando as credenciais do usuário fazendo o pedido. O sistema responde HTTP OK com um arquivo “application/json” contendo todas Agências encontradas. Caso aconteça um erro, HTTP INTERNAL ERROR

OCCURED será indicado e HTTP NOT AUTHORIZED se o usuário não possuir direitos de acesso.

Caso de Uso 1.21 – Autenticar Veículo (CLIENTE MÓVEL / escopo veículo)

O CLIENTE MÓVEL envia, através do HTTP, um arquivo no formato “application/json” para o sistema descrevendo seu identificador e sua senha. O sistema responde HTTP OK caso o usuário possua um papel de veículo. Caso contrário, responde HTTP NOT AUTHORIZED.

3.2.3.2 Casos de Uso da Aplicação CLIENTE MÓVEL

Abaixo seguem os Casos de Uso da aplicação CLIENTE MÓVEL.

Caso de Uso 2.1 – Fazer Check-in (CONDUTOR VEÍCULO / escopo veículo)

O CONDUTOR DO VEÍCULO informa um identificador do veículo que o mesmo está conduzindo e a senha correspondente ao identificador. O sistema responde positivamente, caso o identificador e a senha sejam válidos, ou negativamente, caso os mesmos não sejam válidos.

Caso de Uso 2.2 – Escrever Alerta (CONDUTOR VEÍCULO / escopo veículo)

O CONDUTOR DO VEÍCULO informa a causa, o efeito e uma descrição do Alerta e aperta um botão de ENVIAR. O sistema tentará publicar o Alerta.

3.2.3.3 Casos de Uso da Aplicação CLIENTE WEB

Abaixo estão os Casos de Uso da aplicação CLIENTE WEB.

Caso de Uso 3.1 – Listar Agências (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todas Agências representadas pelos seus nomes, websites e telefones.

Caso de Uso 3.2 – Cadastrar Agência (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o nome, o website, o fuso horário, a língua, o telefone e o website para compra de passagens, e irá submeter esses dados. O sistema avisará o usuário se um dos dados obrigatórios não for informado. O nome, o website, o fuso horário e a língua são dados obrigatórios. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.3 – Listar Rotas (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todas Rotas representadas pelos seus nomes e agências.

Caso de Uso 3.4 – Cadastrar Rota (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o nome curto, nome longo, a descrição, o tipo de rota, o website da rota, a cor da rota e a cor do texto da rota, e irá submeter esses dados. O sistema avisará o usuário se um dos dados obrigatórios não for informado. Os nomes, o tipo de rota e a descrição são dados obrigatórios. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.5 – Listar Viagens (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todas Viagens representadas pelos seus nomes, rotas, disponibilidades de serviço e agências.

Caso de Uso 3.6 – Cadastrar Viagem (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o nome, o headsign no veículo, a direção. a rota e o serviço, e irá submeter esses dados. O sistema avisará o usuário se um dos dados obrigatórios não for informado. Todos os dados são dados obrigatórios. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.7 – Listar Paradas de Veículo (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todas Paradas de Veículo representadas pelos seus nomes, códigos, tipos de parada, latitudes, longitudes e as estações a que pertencem.

Caso de Uso 3.8 – Cadastrar Parada de Veículo (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o nome, o código, a descrição, a latitude, a longitude, o tipo de parada, o fuso horário, o website, se suporta cadeira de rodas, e irá submeter esses dados. O sistema avisará ao usuário se um dos dados obrigatórios não for informado. A latitude e a longitude são dados obrigatórios. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.9 – Listar Veículos (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todos Veículos representados pelos seus nomes, agências, placas de licença e os usuários no sistema que representam os veículos.

Caso de Uso 3.10 – Cadastrar Veículo (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o usuário mapeado para esse veículo, a agência do veículo, o nome e a placa de licença, e irá submeter esses dados. O sistema avisará ao usuário se um dos dados obrigatórios não for informado. O usuário, e a agência são dados obrigatórios. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.11 – Listar Disponibilidades de Serviço (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todas Disponibilidades de Serviço representadas por seus códigos.

Caso de Uso 3.12 – Cadastrar Disponibilidade de Serviço (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o código e datas em que o serviço não estará disponível, e irá submeter os dados. O sistema avisará ao usuário se um dos dados obrigatórios não for informado. O código é o único dado obrigatório. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.13 – Listar Usuários do Sistema (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá visualizar todos Usuários do sistema representadas por seus identificadores e seus papéis no sistema.

Caso de Uso 3.14 – Cadastrar Usuários do Sistema (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar o identificador do usuário, a senha e os papéis do usuário no sistema, e irá submeter os dados. O sistema avisará ao usuário se um dos dados obrigatórios não for informado. Todos dados são obrigatórios. Caso a submissão tenha sido um sucesso, o USUÁRIO DO SISTEMA receberá uma mensagem.

Caso de Uso 3.15 – Autenticar (USUÁRIO DO SISTEMA / escopo administrador)

O USUÁRIO DO SISTEMA irá informar seu identificador no sistema e sua senha, e irá submeter os dados. Se os dados estiverem corretos e o usuário tiver acesso ao sistema, o mesmo será direcionado para a página inicial, caso contrário, uma notificação de erro será feita.

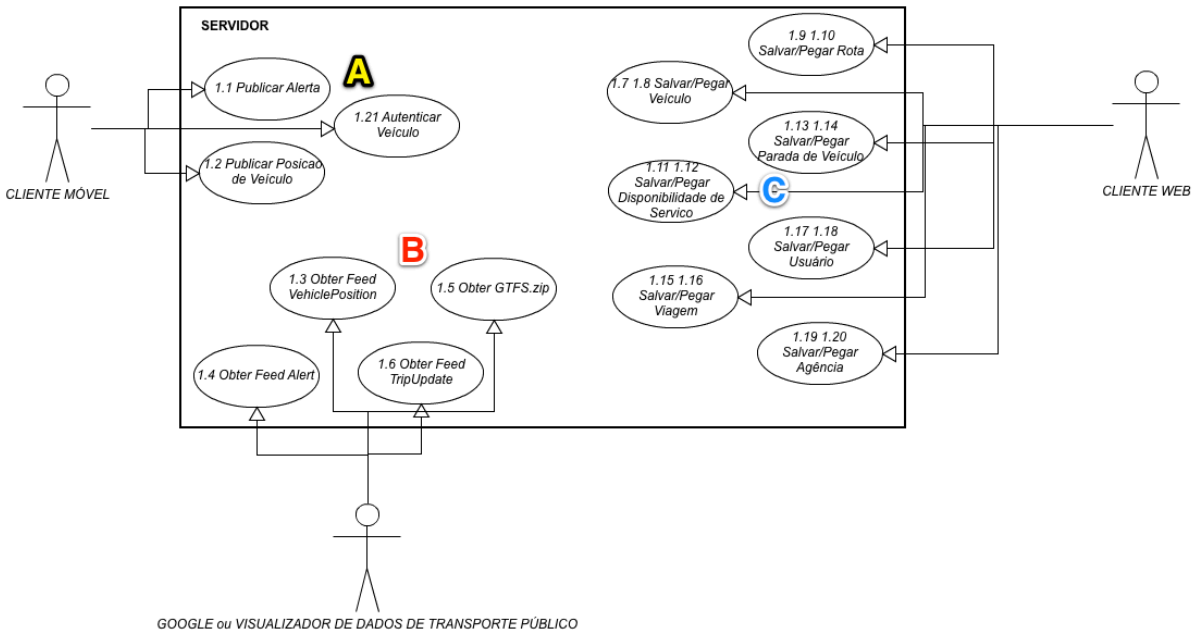
3.2.3.4 Diagramas dos Casos de Uso

Abaixo, pode-se observar o diagrama de Casos de Uso em relação a aplicação SERVIDOR. Existem três atores: CLIENTE MÓVEL, CLIENTE WEB, e GOOGLE OU VISUALIZADOR DE DADOS DE TRANSPORTE PÚBLICO.

Observa-se, na figura, em “A”, o CLIENTE MÓVEL, que é um sistema externo, como um ator pois precisará publicar alertas e publicar a posição atual de um veículo. Já em “B”, observa-se o GOOGLE que irá precisar obter o arquivo descrevendo dados estáticos GTFS, somado aos três arquivos GTFS-realtime que representam as mensagens “VehiclePosition”, “TripUpdate” e “Alert”. Por fim, em “C”, pode-se ver o CLIENTE WEB como o último ator que irá precisar acessar

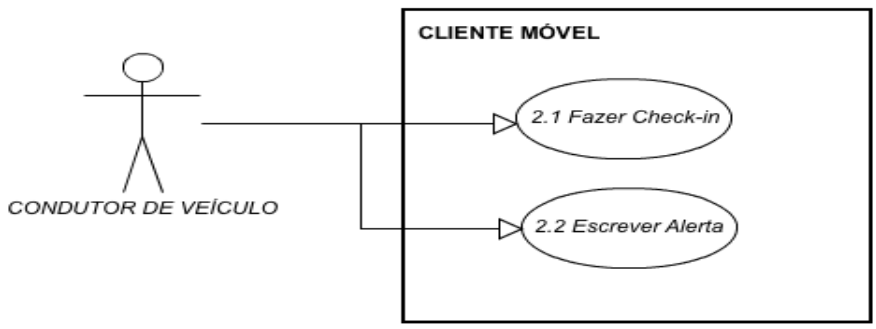
serviços tanto para salvar como para ler Agências, Rotas, Veículos, Usuários, Viagens e Disponibilidade de Serviços.

Figura 22 - Diagrama de Casos de Uso da aplicação Servidor



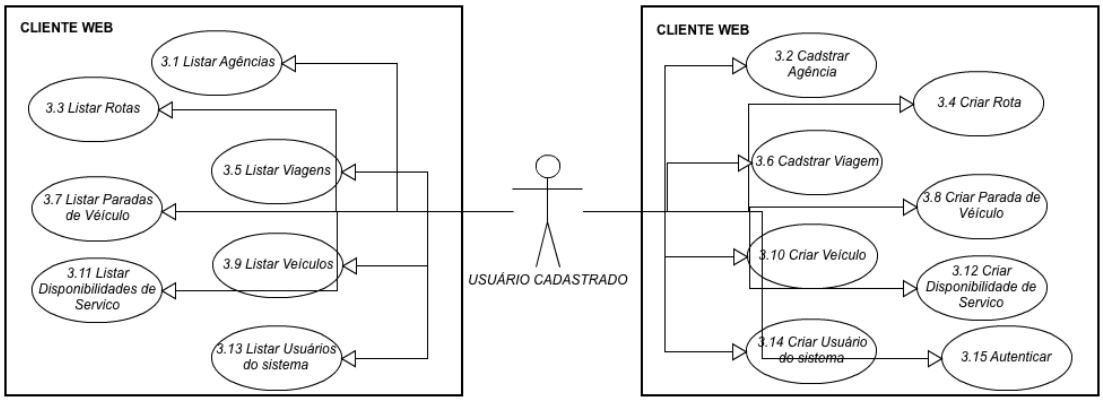
Já nessa outra figura abaixo, podemos observar os dois Casos de Uso da aplicação CLIENTE MÓVEL. Um condutor de um veículo ou alguma outra pessoa responsável precisará fazer um check-in e poderá escrever alertas.

Figura 23 - Diagrama de Casos de Uso da aplicação Cliente Móvel



Por fim, podemos observar na figura abaixo, o diagrama de Casos de Uso da aplicação CLIENTE WEB. Um usuário cadastrado no sistema poderá criar e ver os dados cadastrados no sistema.

Figura 24 - Diagrama de Casos de Uso da aplicação Cliente Web



3.3 Materiais e Métodos do SERVIDOR

Para o desenvolvimento do SERVIDOR que será responsável por armazenar os dados a respeito das empresas de trânsito seguindo o modelo GTFS-realtime, as tecnologias descritas abaixo serão utilizadas.

O Spring Roo será utilizado para desenvolvimento inicial da aplicação, ou seja, para a configuração inicial e geração de código. Será removido, a fim de permitir mais liberdade no desenvolvimento.

O Spring Framework será utilizado para o desenvolvimento da camada de serviço da aplicação, para controlar o acesso ao banco de dados, e para questões relacionadas a segurança como restrição de acessos a determinados recursos da aplicação. Juntamente com esse, será utilizado o QueryDSL para definição das

consultas ao banco de dados. Para gerenciar os serviços web será utilizado o Spring MVC, que é parte do Spring Framework.

Para conversão dos dados armazenados no banco de dados para o padrão GTFS-realtime, serão utilizados módulos do projeto One Bus Away [23], que será descrito em seguida.

3.3.1 Projeto One Bus Away

O projeto One Bus Away [23] é um projeto que possui como objetivo melhorar a usabilidade do transporte público através do desenvolvimento de aplicações para diversos dispositivos que informam dados de transporte público. Além disso, eles provem diversas bibliotecas e aplicativos open-source para auxiliar tanto usuários, como desenvolvedores e como agências de trânsito.

Baseado em uma análise inicial, esse trabalho irá utilizar módulos de conversão de dados para o formato do GTFS.

Para fim de testes, será utilizado o módulo “OneBusAway Web” como um outro cliente responsável por exibir os dados de trânsito em formato similar ao Google Transit, ou seja, com mapas. Já que o Google Transit não possui um sistema para testes, esse trabalho não poderá publicar os dados no sistema do Google, mas, através desse módulo do One Bus Away será possível fazer uma demonstração do sistema.

3.4 Materiais e Métodos do CLIENTE WEB

Para desenvolvimento do CLIENTE WEB será utilizado o GWT, já que o autor está buscando conhecimento nessa tecnologia. Diante do estudo realizado no

framework MVP do GWT e a grande quantidade de trabalho que o mesmo exige para criação de uma aplicação, o framework MVP4G [26] será utilizado para acelerar o desenvolvimento. Também será utilizado o RestyGWT [27], uma biblioteca que gera automaticamente clientes para serviços web.

3.4.1 MVP4G

O framework MVP4G é um poderoso framework que permite que aplicações sejam criadas seguindo os conceitos já vistos pelo framework MVP do GWT 2.4. Esse framework irá simplificar o código, já que todas as configurações necessárias para utilização do MVP é realizada através de uma única classe.

Através da criação de uma classe EventBus da aplicação, eventos, lugares, views e presenters podem ser declarados somente nessa. A grande vantagem disso é a simplificação total do código, caso comparado com o que é gerado pelo Spring Roo para uma aplicação GWT.

Em geral, o MVP4G permite que todos os conceitos de MVP sejam utilizados no GWT, sem que isso interfira com a produtividade do desenvolvedor.

3.5 Materiais e Métodos do CLIENTE MÓVEL

O CLIENTE MÓVEL, que funcionará em veículos, será desenvolvido para o sistema operacional Android, ou seja, será necessário o uso de um aparelho. Essa escolha foi feita devido aos grandes recursos disponíveis em dispositivos Android como, por exemplo, o GPS e o acesso a internet, bem como a facilidade para testar o sistema. O dispositivo do autor será utilizado para testes.

Para desenvolvimento do cliente Android, também será utilizado o Spring Framework para Android, pois possui bibliotecas que facilitarão a comunicação com os serviços web RESTful da aplicação web.

A seguir será descrito como uma aplicação Android é programada.

3.5.1 Android

Android é um sistema operacional de código aberto desenvolvido pelo Google que é baseado no Linux. Para os desenvolvedores de aplicações para Android, existe um kit de desenvolvimento em Java.

Uma aplicação Android tem como principal característica um arquivo de descrição chamado de “AndroidManifest”. Esse arquivo irá descrever qual é a tela inicial da aplicação, quais são as outras telas dessa, se existe um objeto “Application” customizado, e regras de acesso quanto a recursos do sistema operacional como a Internet ou um dispositivo de armazenamento externo.

Normalmente, uma aplicação Android irá possuir um objeto que estende a classe “Application”. Esse objeto será acessível por toda a aplicação e, por isso, é interessante o mesmo carregar configurações globais.

Cada tela de uma aplicação é definida através de uma “Activity”, ou Atividade. Cada Atividade terá um ciclo de vida que será utilizado pelo sistema operacional para gerenciar a mesma. Por exemplo, se o usuário receber uma ligação, a aplicação precisa parar e então o estado da Atividade irá mudar e, para isso, o programador terá que fazer um tratamento para quando isso acontecer.

Para cada Atividade, normalmente, é criado um arquivo XML onde é definido o layout da tela, ou seja, componentes como caixas de texto, listas e etc. Quando existe a necessidade da troca de tela, a Atividade terá que chamar uma próxima

Atividade. O interessante desse processo é que o histórico de Atividades funciona como uma pilha: quando o usuário utiliza o botão VOLTAR do aparelho, o próprio sistema operacional é responsável por voltar para a Atividade anterior sem perder nenhuma informação.

Em geral, uma aplicação Android tem uma arquitetura bem simples. Na minha opinião, o framework para desenvolvimento é excelente e isso pode ser comprovado pelo fato de que os desenvolvedores do GWT copiaram algumas idéias da arquitetura do Android para o GWT, e, por isso, conseguimos observar conceitos similares de Atividades tanto no Android como no GWT.

3.6 Conclusões

Nesse capítulo, apresentou-se o modelo de dados que será utilizado por esse trabalho, bem como qual será o projeto a ser desenvolvido. Em geral, foram apresentados os seguintes itens:

Modelo de dados relacional proposto por esse trabalho para o GTFS-realtime.

Projeto a ser desenvolvido por esse trabalho, que consiste em três aplicações: o SERVIDOR, armazenar, receber e transmitir dados no formato GTFS-realtime, o CLIENTE WEB, que é uma aplicação para visualização e produção de dados do SERVIDOR, e o CLIENTE MÓVEL, que é uma aplicação que será executada em um veículo de trânsito e responsável por transmitir dados como a posição do veículo para o SERVIDOR.

Os métodos e materiais a serem utilizados no projeto: Spring Framework no SERVIDOR, GWT no CLIENTE WEB e Android no CLIENTE MÓVEL.

4 DESENVOLVIMENTO DO PROJETO PROPOSTO

Nesse capítulo será detalhado como o desenvolvimento do projeto proposto no capítulo 3 foi feito. O primeiro passo é o desenvolvimento do SERVIDOR, já que o CLIENTE WEB e o CLIENTE MÓVEL precisam acessar dados armazenados no SERVIDOR através de serviços web.

4.1 Desenvolvimento do SERVIDOR

O SERVIDOR será separado em três projetos Java: CORE, API e API-COMMON. O primeiro projeto será o CORE e nele estarão todas as classes necessárias para acesso ao banco de dados e a lógica de negócio, ou seja a camada de serviço. Em seguida, no projeto API serão geradas as classes que serão o ponto de entrada do mundo externo para a aplicação, ou seja, basicamente, serviços web. Por fim, o projeto API-COMMON representará classes que podem ser compartilhadas entre todos os projetos.

4.1.1 Desenvolvimento do projeto Java CORE

O projeto CORE será o responsável por conter todas as classes para acesso ao banco de dados e a lógicas de negócio. A arquitetura SOA será feita nesse projeto, com a geração dos serviços e repositórios. Será utilizado o Spring Roo para geração dos componentes da arquitetura.

Com o modelo relacional definido no capítulo 3, o primeiro passo é criar o banco de dados. Através da ferramenta MYSQL Workbench [28], pode-se, diante de um modelo relacional, criar um banco de dados. Em seguida, através do Spring Roo,

é possível gerar as classes das entidades do modelo em Java por meio do comando “database reverse engineer” exibido abaixo.

database reverse engineer --schema no-schema-required --package ~.server.domain --activeRecord false

Ou seja, ao invés de gerar o banco de dados a partir do Spring Roo, como apresentado no 2.3, já que o banco de dados existe, é mais fácil utilizar esse comando do Spring Roo. Serão economizadas centenas linhas de comandos, já que descrever novamente todas as entidades e atributos com comandos do Roo seria um exagero. Através desse comando, classes Java com anotações seguindo a especificação JPA serão criadas.

Um próximo passo é a geração da camada de serviço da aplicação segundo a arquitetura SOA. Para isso, como já visto no 2.3, será utilizado o Spring Roo para gerar repositórios e serviços configurados com o Spring Framework. Serão necessários dois comandos por entidade do modelo relacional. Abaixo está um exemplo desses dois comandos para a criação desses elementos para a entidade “Agency”:

repository jpa --interface ~.server.repository.AgencyRepository --entity ~.server.domain.Agency
repository jpa --interface ~.server.service.AgencyService --entity ~.server.domain.Agency

Portanto, rapidamente, pode ser gerado todas as classes necessárias para as operações CRUD [30] com o Spring Roo. Depois disso, a remoção do Spring Roo

pode ser feita através do IDE SpringSource Tool Suite. Já que a biblioteca QueryDSL será utilizada, a mesma deverá ser configurada manualmente para cada entidade da mesma forma descrita no capítulo 2.

O Spring Security deverá ser configurado nesse projeto, ou seja, a classe UserDetailsService precisará ser implementada. Isso pode ser feito, fazendo um dos serviços gerados pelo Roo ser superclasse dessa. Tendo a implementação, basta a configuração em um arquivo “applicationContext” no formato XML da mesma forma como foi visto no capítulo 2.

Por fim, métodos de serviços necessários pelos casos de uso deverão ser criados e implementados manualmente. Isso será feito apenas no projeto API, já que todos os casos de uso do SERVIDOR são relacionados a serviços web.

4.1.2 Desenvolvimento do projeto Java API e API-COMMON

O projeto API será responsável por ser o ponto de entrada de pedidos externos em relação aos dados armazenados no banco de dados. Como já foi visto no capítulo 2, é fácil a implementação de serviços web REST através do Spring Framework. Esse projeto terá como dependência o projeto CORE.

O CLIENTE WEB precisará acessar diversos recursos definidos no capítulo 3. Por isso, o ideal é a criação de um Controlador Spring MVC para cada caso de uso ou para cada grupo de caso de uso relacionado. Já que os serviços web REST utilizarão o JSON como meio serialização de dados, existe a necessidade da criação de objetos POJOS [31].

POJOS, ou “Plain Old Java Objects”, são objetos Java que não possuem nenhuma restrição ou dependência de bibliotecas. Existe a necessidade da utilização desses, pois, por exemplo, caso uma Agência seja serializada do jeito que

ela está modelada no modelo entidade relacionamento, o banco de dados inteiro seria transmitido devido ao relacionamento de uma Agência com muitas entidades. A melhor estratégia em termos de performance é a criação de um POJO por caso de uso. Nesse projeto, foi decidido criar apenas um POJO por entidade no modelo entidade relacionamento.

O diagrama de classes dos POJOS será exibido na análise de cada caso de uso, já que a determinação de que relacionamento cada POJO irá possuir dependerá dos casos de uso. O padrão para nomeação das classes foi “NomeDaEntidadeJSONObject”, já que o POJO será serializado diretamente para JSON.

Os POJOS serão colocados no projeto API-COMMON, pois, dessa forma, é possível a utilização desses objetos nos projetos CLIENTE WEB e CLIENTE MÓVEL. Basicamente o API-COMMON será uma dependência em todos os projetos Java, carregará, além de POJOS, ENUMS e outras constantes que possam ser compartilhadas por todos projetos.

Os serviços web REST serão descritos através da WADL [32], ou WEB APPLICATION DESCRIPTION LANGUAGE. Em geral, o WADL é um arquivo no formato XML que descreve serviços web como recursos.

Abaixo então, serão feitas análises de cada caso de uso e como os mesmos serão implementados.

4.1.2.1 Casos de Uso Escopo Veículo

Para o caso de uso 1.1 PUBLICAR ALERTA, é necessário a criação de um recurso REST chamado “AlertResource”. Esse recurso pode ser descrito através do

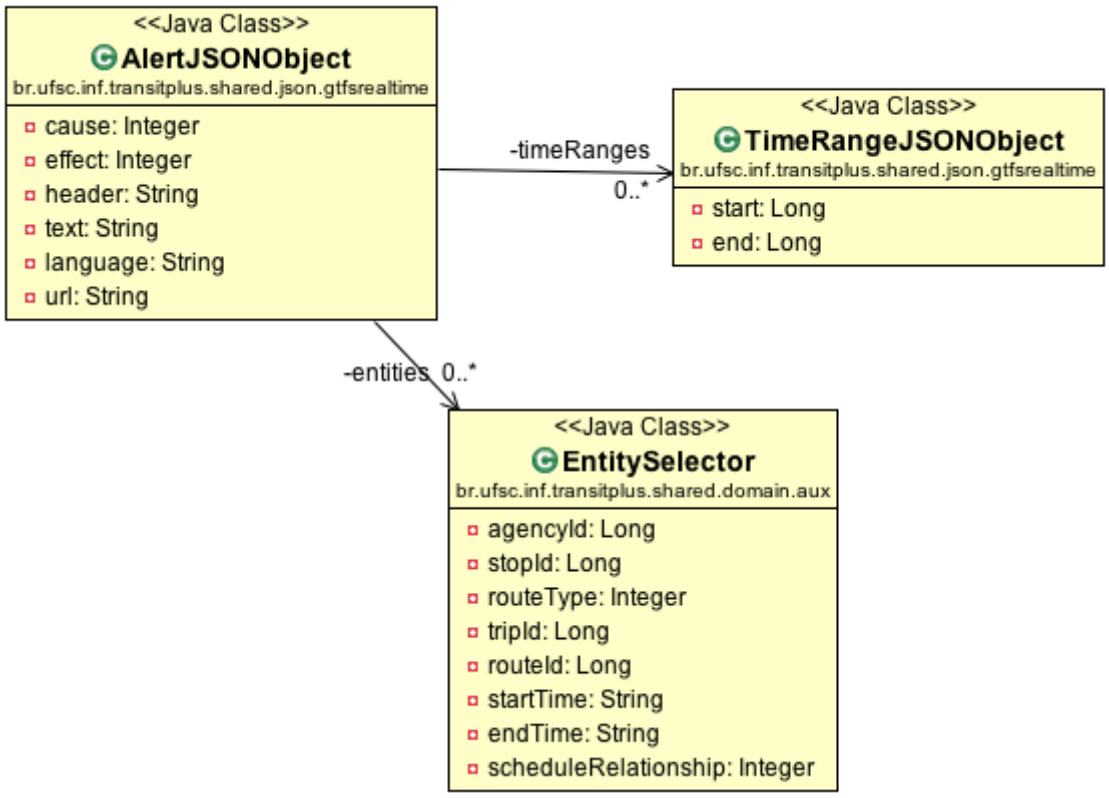
WADL através da imagem abaixo. Ele receberá um HTTP POST contendo o POJO “AlertJSONObject” serializado como JSON.

Figura 25 - Serviço-web da entidade Alert descrito através da WADL

```
<resource path="/realtime/alert">
  <method id="publishAlert" name="POST">
    <doc title="br.ufsc.inf.transitplus.server.rest.resource.realtime.AlertResource.publishAlert"/>
  </method>
</resource>
```

Na mensagem ALERT do GTFS-realtime, é possível indicar mensagens internacionalizadas. Nesse Caso de Uso, já que o Alerta vem direto de um veículo, imagina-se que apenas uma única mensagem na língua da Agência será publicada. Por isso, o POJO do Alerta receberá apenas uma única versão para a mensagem. Um mensagem ALERT possui ainda mensagens do tipo TIME RANGE. No caso do POJO, um condutor poderá selecionar mais de data de início e fim de uma Alerta. Também poderá informar para quais escopos esse Alerta possui através do POJO “EntitySelector”.

Figura 26 - POJO da entidade Alert

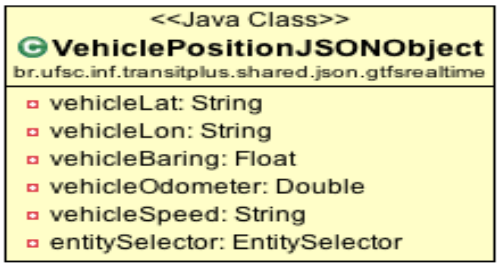


Já para o Caso de Uso 1.2 PUBLICAR posição DE VEÍCULO, também é necessário a criação de um recurso REST chamado de “VehiclePositionResource”. Esse recurso pode ser descrito através do WADL através da imagem abaixo. Ele receberá um HTTP POST contendo o POJO “VehiclePositionJSONObject” serializado como JSON.

Figura 27 - Serviço-web da entidade Vehicle Position descrito através da WADL

```
▼<resource path="/realtime/vehicleposition//publish">
  ▼<method id="publishVehiclePosition" name="POST">
     <doc title="br.ufsc.inf.transitplus.server.rest.resource.realtime.VehiclePositionResource.publishVehiclePosition"/>
   </method>
 </resource>
```

Figura 28 - POJO da entidade Vehicle Position



O POJO “VehiclePositionJSONObject” conterá todos os valores suportados pelo GTFS-realtime VEHICLE POSITION a respeito do estado atual de um veículo.

Em relação a esses dois Casos de Uso, em termos de implementação, é necessário implementar uma validação dos dados, ou seja, se o POJO contém todos os dados obrigatórios e um método de transformação de POJO para ENTIDADE do modelo entidade relacionamento. A validação pode ser implementada através da biblioteca Hibernate Validator [31], que utiliza anotações no objeto Java para validar o mesmo.

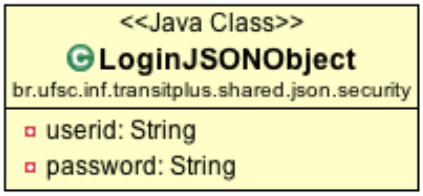
Por fim, o último Caso de Uso no escopo VEÍCULO é o 1.21 AUTENTICAR VEÍCULO. Para esse, é necessário a criação de um novo POJO “LoginJSONObject”, que irá conter o usuário e a senha de uma tentativa de autenticação. Um novo

recurso REST é criado através da classe “VehicleAutenticationResource”. Abaixo está a descrição WADL e o POJO.

Figura 29 - Serviço-web para autenticação de veículos descrito através da WADL

```
▼<resource path="/vehicle//login">
 ▼<method id="login" name="POST">
    <doc title="br.ufsc.inf.transitplus.server.rest.resource.vehicle.VehicleAuthenticationResource.login"/>
  </method>
 </resource>
```

Figura 30 - POJO para autenticacao de veículo

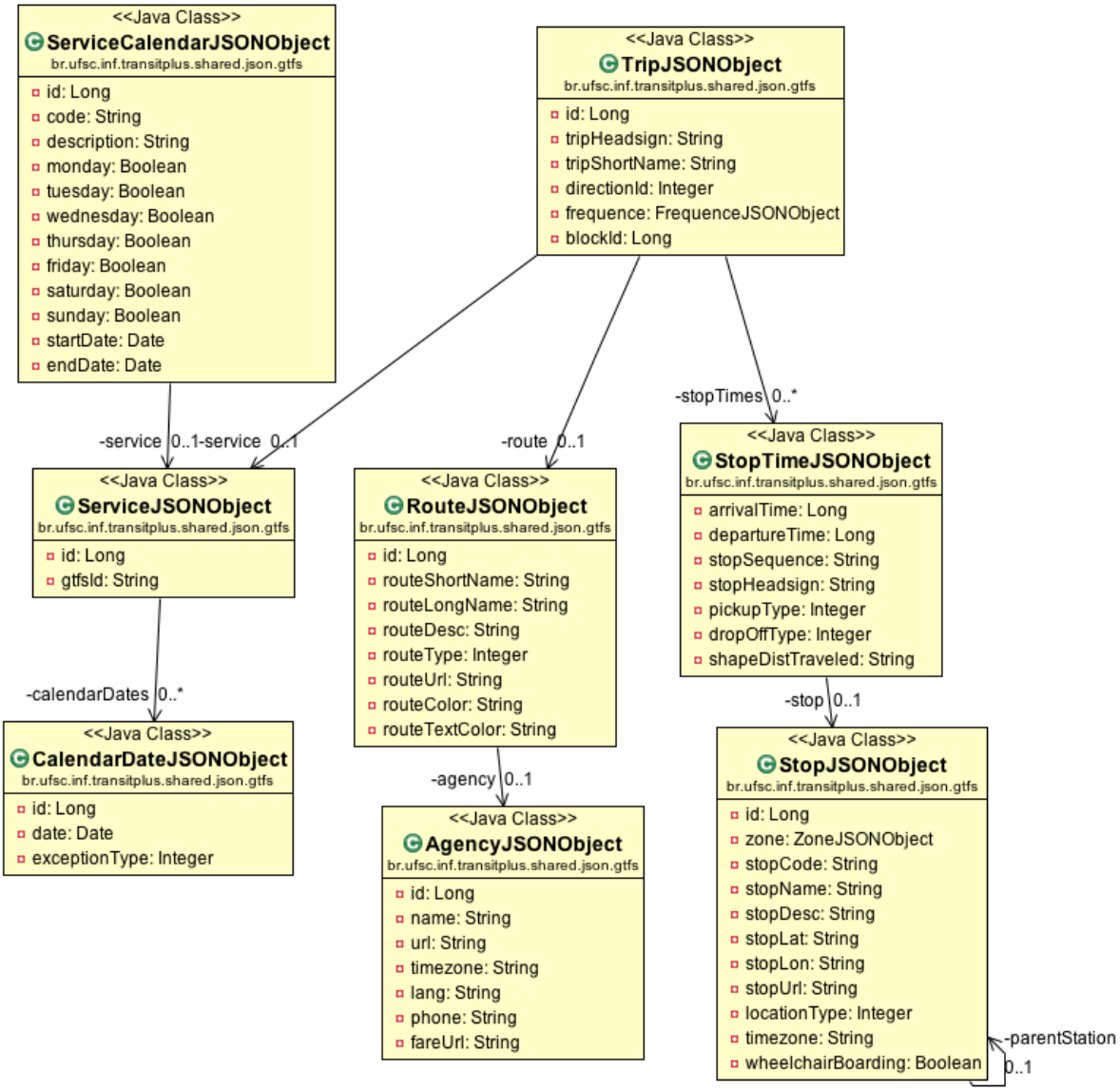


Para implementação desse serviço é necessário a verificação se o identificador de usuário enviado existe, está ativo e possui o papel de veículo. Também é necessário a utilização do objeto “AuthenticationManager” que é do framework Spring Security para tentar uma autenticação.

4.1.2.2 Casos de Uso Escopo Serviços

Esses Casos de Uso do escopo Serviços representam operações de criação e listagem de dados no banco de dados. É necessário a criação de POJOS, conforme referido anteriormente. Para demonstrar a implementação desses Casos de Uso, será exibido um diagrama contendo todos os POJOS envolvidos e um exemplo será discutido. Abaixo está o diagrama desses novos POJOS:

Figura 31 - POJOs para entidades que representam o GTF



Analisando os Casos de Uso 1.19 e 1.20, SALVAR AGÊNCIA e PEGAR AGÊNCIAS respectivamente, é necessário a criação de uma chamada HTTP para cada Caso de Uso. O padrão a ser adotado por esse projeto para esses Casos de Uso é a criação de uma classe para cada ENTIDADE.

Especificamente quanto a esse caso, cria-se uma única classe “AgencyServiceResource” que contém dois recursos REST, um para salvar a entidade, outro para obter entidades. Na verdade, nesse caso serão três recursos REST, pois, em geral, no Caso de Uso PEGAR AGÊNCIAS, serão necessários dois

recursos REST: um para obter as Agências, e outro para contar o total de Agências. Abaixo está o WADL:

Figura 32 - Serviços-web para CRUD da entidade Agency descrito através da WADL

Em geral, todos os Casos de Uso desse escopo seguirão o mesmo padrão. De certa forma, poderiam ter sido gerados automaticamente esses recursos REST, se tivesse uma maneira de especificar que atributos de uma entidade no modelo entidade-relacionamento fariam parte do POJO. O Spring Roo suporta a geração automática de recursos REST, mas, nesse caso, não faria muito sentido diante do problema da serialização: a entidade Agência, por exemplo, tem relação com todas as outras entidades praticamente e isso causaria problemas de performance na serialização.

4.1.2.3 Casos de Uso Escopo Visualizadores de Dados

Nesse escopo, o objetivo é simplesmente disponibilizar dados em formatos pré-determinados como recursos REST.

4.1.2.3.1 Caso de Uso 1.5 Obter GTFS.zip

Para o Caso de Uso 1.5 Obter GTFS.zip, o objetivo é disponibilizar dados estáticos de uma agência de trânsito, ou seja, viagens, paradas de ônibus,

disponibilidade de seus serviços em um formato estabelecido GTFS. Dados GTFS, conforme visto em JORGE (2011), precisam ser descritos em simples arquivos de texto, onde, a primeira linha representa os nomes separados por vírgula dos dados a serem descritos, e as linhas seguintes representam diferentes entradas desses dados. Abaixo está um exemplo do arquivo “agency.txt”:

Figura 33 - Exemplo do arquivo agency.txt do GTFS

```
agency_id, agency_name,agency_url,agency_timezone,agency_phone,agency_lang,agency_fare_url
1,MBTA,http://www.mbta.com,America/Sao_Paulo,617-222-3200,EN,
```

Para disponibilizar todos dados, é necessária a criação de uma Classe Java para gerar todos os arquivos TXT esperados pelo GTFS para cada Agência cadastrada no sistema. Além disso, é necessário a criação de um recurso REST para responder com o arquivo.

A estratégia utilizada para disponibilização desses arquivos GTFS.zip, é a criação de uma tarefa de tempos em tempos. Por exemplo, uma vez por dia, pode-se processar todos os dados de um agência e deixar os arquivos GTFS.zip por agência prontos. Isso pode ser feito facilmente, já que o projeto Java é feito com o framework Spring, e o mesmo possui suporte para esse tipo de tarefa agendada. Mais informações podem ser encontradas em “Task Execution and Scheduling” [33].

Por fim, é criado um recurso REST que responde a pedidos pelo GTFS.zip de uma agência especificada na URL do pedido HTTP. Abaixo está a imagem do WADL desse recurso.

Figura 34 - Serviço-web para disponibilização do arquivo GTFS.zip descrito através da WADL

```
▼<resource path="/feed/gtfs/{agencyId}">
  ▼<method id="getFile" name="GET">
     <doc title="br.ufsc.inf.transitplus.server.rest.resource.exporter.GtfsExporter.getFile"/>
    ▼<request>
       <param name="agencyId" style="template" required="true"/>
     </request>
   </method>
</resource>
```

4.1.2.3.2 Casos de Uso Realtime

Os outros três Casos de Uso desse escopo são relativos as três mensagens do GTFS-realtime, que, por serem mensagens que devem ser constantemente atualizadas, precisam de uma estratégia um pouco diferente. A segunda grande diferença entre esses Casos de Uso e o 1.5 é o formato de dados: esses Casos de Uso precisam ser disponibilizados em um recurso REST no formato "application/x-google-protobuf" que é um formato feito pelo Google.

Para facilitar a implementação desses novos recursos REST, será utilizada a biblioteca “onebusaway-gtfs-realtime-exporter” [34]. Por meio dessa biblioteca, é possível exportar dados para o formato Google PROTOBUFFER sem conhece-lo. Através da classe “FeedMessage”, pode-se adicionar todos os dados na mesma e chamar o método “FeedMessage.writeTo”, passando o stream da resposta HTTP, para escrever os dados no formato proposto pelo Google.

A grande diferença para o Caso de Uso 1.5 é que esses dados, em teoria, precisam ser atualizados a todo momento. Uma estratégia possível é, a cada 1 minuto, por exemplo, atualizar essas mensagens GTFS-realtime. Essa foi a estratégia escolhida.

Por fim, são criados então, três novos recursos REST, um para cada Caso de Uso, sendo que é requerido o identificador da Agência para acessar esses. Abaixo está o WADL desses novos, incluindo o apresentado anteriormente do Caso de Uso 1.5.

Figura 35 - Serviços-web para disponibilização dos feeds GTFS-realtime descrito através da WADL

```
▼<resource path="/feed/alert/{agencyId}">
 ▼<method id="get" name="GET">
    <doc title="br.ufsc.inf.transitplus.server.rest.resource.exporter.ExporterBase.get"/>
  ▼<request>
     <param name="" style="template" required="true"/>
     <param name="" style="query" default="false" required="false"/>
   </request>
  ▼<response status="200">
     <representation mediaType="application/x-google-protobuf"/>
   </response>
 </method>
</resource>
▼<resource path="/feed/gtfs/{agencyId}">
 ▼<method id="getFile" name="GET">
    <doc title="br.ufsc.inf.transitplus.server.rest.resource.exporter.GtfsExporter.getFile"/>
  ▼<request>
     <param name="agencyId" style="template" required="true"/>
   </request>
 </method>
</resource>
▼<resource path="/feed/trip-update/{agencyId}">
 ▼<method id="get" name="GET">
    <doc title="br.ufsc.inf.transitplus.server.rest.resource.exporter.ExporterBase.get"/>
  ▼<request>
     <param name="" style="template" required="true"/>
     <param name="" style="query" default="false" required="false"/>
   </request>
  ▼<response status="200">
     <representation mediaType="application/x-google-protobuf"/>
   </response>
 </method>
</resource>
▼<resource path="/feed/vehicle-position/{agencyId}">
 ▼<method id="get" name="GET">
    <doc title="br.ufsc.inf.transitplus.server.rest.resource.exporter.ExporterBase.get"/>
  ▼<request>
     <param name="" style="template" required="true"/>
     <param name="" style="query" default="false" required="false"/>
   </request>
  ▼<response status="200">
     <representation mediaType="application/x-google-protobuf"/>
   </response>
 </method>
</resource>
```

4.1.2.4 Protegendo Recursos REST

Todos os Casos de Uso do SERVIDOR, com exceção dos Casos de Uso com escopo VISUALIZADOR DE DADOS, precisarão de proteção. Ou seja, apenas determinados usuários terão acesso aos recursos. Conforme já visto no capítulo 2, não é difícil a configuração do Spring Security para recursos REST. Abaixo está a imagem da configuração utilizada no projeto API.

O Spring Security utiliza a configuração de interceptadores de URL para determinar o nível de acesso de cada uma. Quando um pedido é feito, uma URL é

testada contra todas as configurações de interceptação definidas. Por isso, a ordem em que são colocados os interceptadores é de extrema importância.

Os únicos serviços que serão abertos a qualquer pedido são os feitos aos recursos dos Casos de Uso com escopo VISUALIZADOR DE DADOS e isso pode ser observado na configuração, que possui um interceptador em todas URLs com formato “api/feed/**”.

A outra exceção é o Caso de Uso 1.21 AUTENTICAR VEÍCULO, onde a autorização é feita no próprio código Java, devido a verificações extras que devem ser feitas. Em teoria, o Spring Security suporta múltiplos níveis de teste de autorização, mas, nesse caso, foi mais rápido a implantação direto por Java.

A última configuração de interceptação é para dizer que todas as URLs irão requerer que um usuário esteja autenticado.

Figura 36 - Arquivo de configuração do Spring Security para proteção dos servicos-web

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.1.xsd">

    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <!-- PRIMARY CONFIG -->
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

    <!-- This is the primary Spring Security Configuration -->

    <http create-session="stateless" pattern="/**" use-expressions="true">
        <intercept-url pattern="/api/feed/**" access="permitAll" />
        <intercept-url pattern="/api/vehicle/login" access="permitAll" />
        <intercept-url pattern="/**" access="isAuthenticated()" />
        <http-basic />
    </http>

    <global-method-security secured-annotations="enabled" />

</beans:beans>
```

A definição GLOBAL-METHOD-SECURITY irá dizer que é possível definir restrições de acesso dentro de métodos. Isso irá oferecer uma maior facilidade de configuração. Pode-se então, restringir o acesso a todos os recursos REST dos Casos de Uso com escopo Serviço, por meio de uma anotação Java “@Secured”.

4.1.3 Conclusões sobre Desenvolvimento do SERVIDOR

Nesse subcapítulo foi descrito como o SERVIDOR foi implementado.

Utilizou-se a subdivisão de três projetos Java: CORE, API e API-COMMON.

CORE representa acesso a camada de serviço e banco de dados.

API caracterizou-se por implementar todos os Casos de Uso do SERVIDOR por meio de webservices.

API-COMMON caracterizou-se por compartilhar todos os POJOS entre todos os projetos.

4.2 Desenvolvimento do CLIENTE WEB

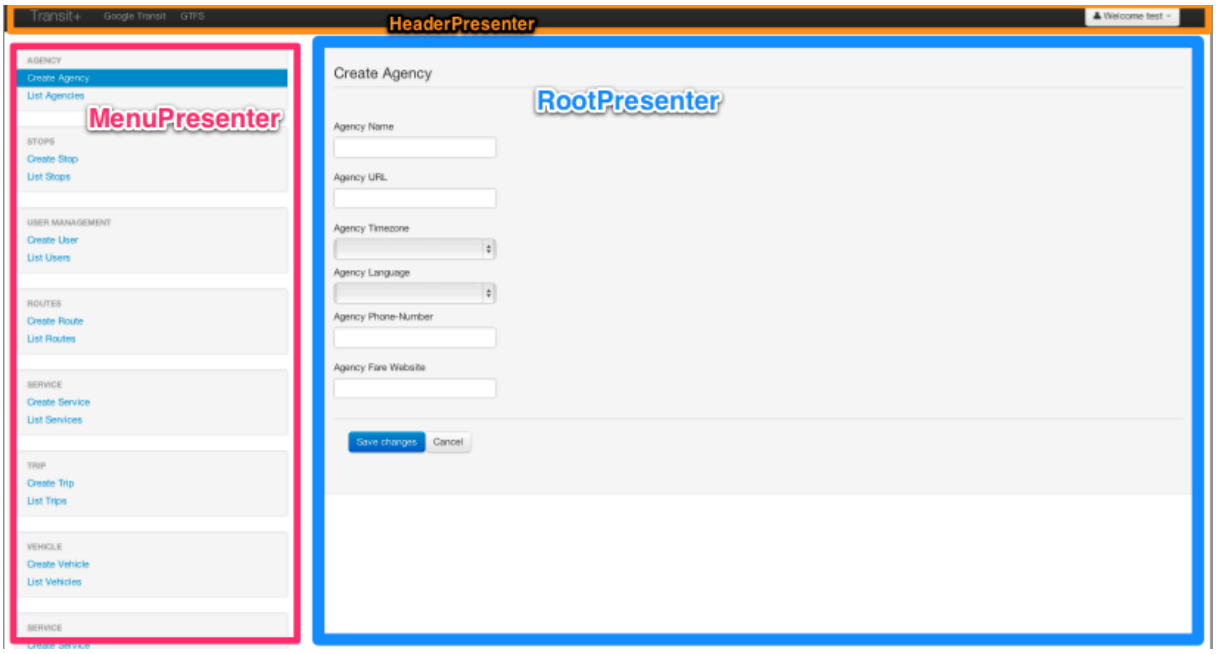
O CLIENTE WEB é o cliente responsável por gerenciar dados armazenados no SERVIDOR através de uma interface gráfica. Em geral, esse cliente irá determinar dados GTFS, já que os dados GTFS-realtime serão gerados por veículos. Além da biblioteca do próprio projeto API-COMMON, como já visto nos requisitos, esse cliente será desenvolvido com GWT, e, como já visto nos métodos e materiais, os frameworks MVP4G e RestyGWT serão utilizados.

Devido a utilização do framework MVP4G, a arquitetura do CLIENTE WEB não foi difícil de implementar, caso seja comparado com o nível de dificuldade do framework padrão do GWT 2.4. A aplicação foi dividida em três grandes partes: menu, topo e o corpo. O menu é representado pelo MenuPresenter, o topo pelo HeaderPresenter e o corpo pelo RootPresenter. Abaixo está uma ilustração da aplicação.

Os Presenters do menu, topo e do corpo são sempre o mesmo, mas o conteúdo do corpo está sempre mudando. Quando o usuário acessa a página

especifica de criação de agências, por exemplo, “/create-agency”, o framework MVP4G detecta um evento de navegação, e isso provoca a criação do Presenter que controla a criação de agências. Esse Presenter, então, será o responsável em enviar um evento para o RootPresenter, informando que o corpo da página precisa ser modificado.

Figura 37 - Aplicação Cliente Web em GWT e sua arquitetura MVP



4.2.1 Desenvolvimento dos Casos de Uso

Para cada um dos Casos de Uso do CLIENTE WEB, foram criados uma dupla de Presenter e View, que são acessados através de seu próprio URL. Para acesso a dados armazenados no SERVIDOR, é utilizada a biblioteca RestyGWT para geração automática de clientes de recursos REST.

Por exemplo, o Caso de Uso 3.1 LISTAR AGÊNCIAS, precisou da implementação de quatro classes. Primeiramente, já que se precisa acessar a

recursos REST, foi declarado o cliente a ser gerado pelo RestyGWT. Abaixo está o código dessa declaração feita em uma Interface Java.

A outra classe a ser implementada é o “ListAgenciesPresenter”, que é o Presenter desse Caso de Uso, e é acessado pela URL “/list-agencies”. Abaixo pode ser observado os dois métodos mais importantes dessa classe. O método “bindView” é chamado quando uma instância do Presenter é criada, ou seja, aqui, é feita a associação entre um objeto de View e o Presenter: configuração de “listeners” de eventos que vem da View. Já o método “onGoToListAgencies” é o método acionado quando o evento de navegação para a página “/list-agencies” é acionado. Ou seja, quando o usuário entrar na página para visualizar todas Agências cadastradas no sistema, esse método será chamado e, o objetivo desse método será enviar um evento para o menu dizendo que a função ativa é essa e um evento para o corpo dizendo que a View que deve ser exibida na tela é a View controlada por esse Presenter.

Uma outra classe implementada é o “AgencyDataProvider”, que é a classe responsável por buscar dados dos recursos REST. Esse provedor de dados precisará saber qual instância de tabela ele deve enviar dados e, por isso, pode-se ver no código acima que a tabela sendo exibida na View é adicionada ao provedor.

Por fim, a última classe a ser implementada é a própria View desse Presenter chamada de “ListAgenciesView”.

Em geral, todos os Casos de Uso passaram por esse processo de criação de três a quatro classes. No caso dos Casos de Uso similares ao Caso de Uso 3.2 CRIAR AGÊNCIA, será necessário a criação de uma classe “AgencyEditor” que é responsável por mapear dados informados pelo usuário a um objeto da classe “AgencyJSONObject”.

4.2.2 Conclusões do Desenvolvimento do CLIENTE WEB

Caso o código desenvolvido para o CLIENTE WEB seja comparado com o código que é gerado pelo Spring Roo para aplicações GWT, visto no capítulo 2, pode-se observar uma diferença muito grande na quantidade de código. O Spring Roo gera código a partir do framework padrão do GWT 2.4, o que dificulta muito, tanto o entendimento, como a customização do código. Por isso, foi decidido a utilização do framework MVP4G. Em geral, nesse capítulo, foi visto:

O framework MVP4G proporcionou um desenvolvimento mais rápido e fácil do que a customização de um código gerado automático pelo Spring Roo.

A biblioteca RestyGWT produz clientes de recursos REST de forma satisfatória, já que se pode acessar aos recursos REST do SERVIDOR sem muitos problemas em um curto tempo, sem a necessidade de codificação de clientes.

4.3 Desenvolvimento do CLIENTE MÓVEL

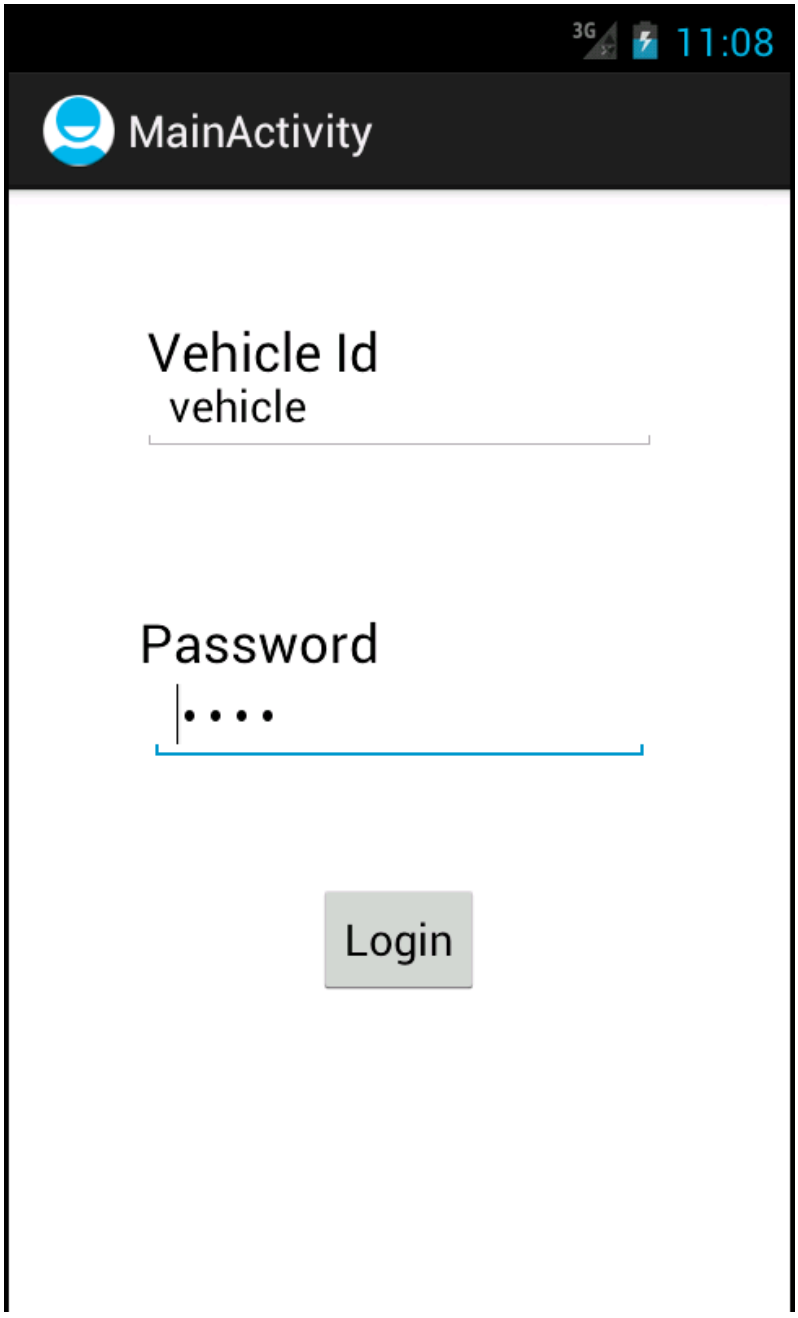
O CLIENTE MÓVEL é aquele cliente responsável por transmitir dados em tempo real de veículos. Como já visto anteriormente, um dos requisitos é a utilização do Android para desenvolvimento do mesmo. Além disso, também já foi visto que será utilizado a biblioteca Spring RestTemplate para acessar recursos REST.

4.3.1 Desenvolvimento dos Casos de Uso

Esse cliente possui apenas dois Casos de Uso. FAZER CHECK-IN e ESCREVER ALERTAS.

O primeiro Caso de Uso, 2.1, provoca o inicio da transmissão da posição do veículo para o SERVIDOR. Para implementação desse, bastou a criação de uma tela de autenticação em que é pedido o identificador do veículo e a senha correspondente. Abaixo está a imagem da aplicação.

Figura 38 - Tela de autenticação da aplicação Cliente Móvel desenvolvida com Android



Já em relação ao segundo Caso de Uso, 2.2, esse não será implementado. Mesmo que não apresente nenhum desafio ou complicações em relação a tempo de desenvolvimento, os testes do Projeto já podem ser feitos sem a implantação desse Caso de Uso.

4.3.2 Desenvolvimento como ATOR do Caso de Uso 1.2

Quando um condutor do veículo faz um CHECK-IN, a aplicação deve ser o ator do Caso de Uso 1.2, PUBLICAR Posição DO VEÍCULO. Para o desenvolvimento disso, o Android oferece APIs internas para acessar a posição atual do veículo obtida junto a um GPS. Utilizando isso, facilmente, pode-se criar uma aplicação que, cada vez que a posição do GPS é modificada, a mesma é transmitida para o SERVIDOR.

Na imagem abaixo, pode ser observado que, a própria API do Android irá informar a aplicação quando uma mudança acontecer na localização atual do aparelho. Quando isso ocorrer, será executada a tarefa de enviar a localização atual para o SERVIDOR.

Figura 39 - Código da aplicação Cliente Móvel para obtenção de dados do GPS

```
// Acquire a reference to the system Location Manager
LocationManager locationManager;

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        // Called when a new location is found by the network location provider.
        new UpdateVehicleLocationTask().execute(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {
    }

    public void onProviderEnabled(String provider) {
    }

    public void onProviderDisabled(String provider) {
    }

};

private class UpdateVehicleLocationTask extends AsyncTask<Location, Void, Void> {

    @Override
    protected Void doInBackground(Location... params) {
        ((TransitVehicleApp) getApplication()).updateVehicleLocation(params[0]);
        return null;
    }

}
```

4.3.3 Conclusões do Desenvolvimento do CLIENTE MÓVEL

O Android possui uma API que facilita muito o acesso a informações como a localização do aparelho. Isso permitiu o desenvolvimento rápido de uma aplicação que informa toda mudança de posição do aparelho. Obviamente talvez essa opção não seja a ideal em termos de custo/benefício, mas considerando que esse trabalho não tem enfoque em hardware, essa aplicação será muito útil para testes iniciais do projeto.

4.4 Conclusões do Desenvolvimento

Nesse capítulo, pode-se mostrar parte do desenvolvimento dos componentes principais do projeto proposto. A respeito do desenvolvimento em geral:

O Spring Framework facilitou muito o desenvolvimento, já que possui soluções compatíveis para diversos requisitos como: tarefas agendadas, segurança e webservices.

O MVP4G também facilitou muito o desenvolvimento, já que simplifica a arquitetura MVP e permite que o programador foque no que é importante e não em questões estruturais do GWT 2.4.

Com o GWT é possível criar aplicações web rapidamente, mas é preciso ter cuidado porque não oferece a oportunidade de customização que um HTML puro ofereceria.

O Android possui boas APIs para acesso a recursos do dispositivo móvel, além de farta documentação.

5 TESTES E CONCLUSÕES FINAIS

Nesse capítulo, serão apresentados testes de componentes específicos e no sistema como um todo. Em seguida, as considerações finais a respeito do trabalho serão feitas.

5.1 Métodos e Materiais dos Testes

Para que seja possível testar o sistema desenvolvido, é necessário a criação de um cenário em que uma agência de transporte público esteja cadastrada no sistema com todos

os dados requeridos. Isso permitirá que os principais Casos de Uso possam ser testados. Além desses, testes de integração serão feitos.

5.1.1 Testes por Caso de Uso

Para testar o sistema como um todo, nos testes de integração, é necessário que alguns Casos de Uso estejam em funcionamento. Abaixo serão listados os Casos de Uso que serão testados.

2.1 Fazer Check-in

1.21 Autenticar Veículo

1.2 Publicar Posição Veículo

1.5 Obter GTFS.zip

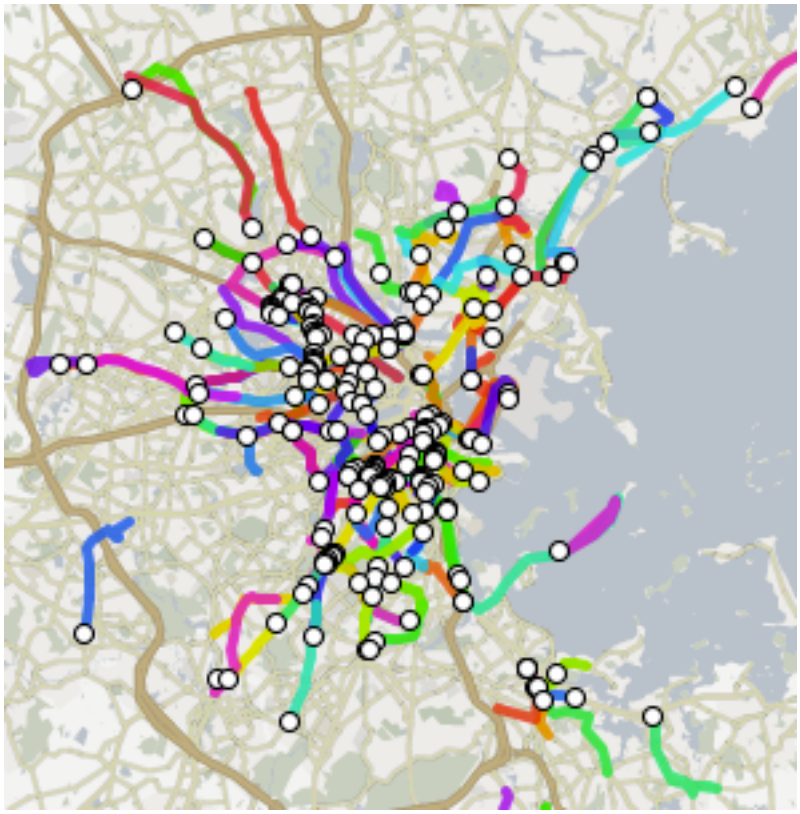
1.3 Obter Feed VehiclePosition

5.1.2 Testes de Integração

Para que o sistema como um todo possa ser testado, será necessário a utilização de aplicações externas que possuam a capacidade de ler os dados no formato do GTFS e GTFS-realtime. Por isso, serão utilizadas as seguintes aplicações desenvolvidas como parte do projeto OneBusAway [23]: Realtime Visualizer [35] e Quickstart Bundle [36].

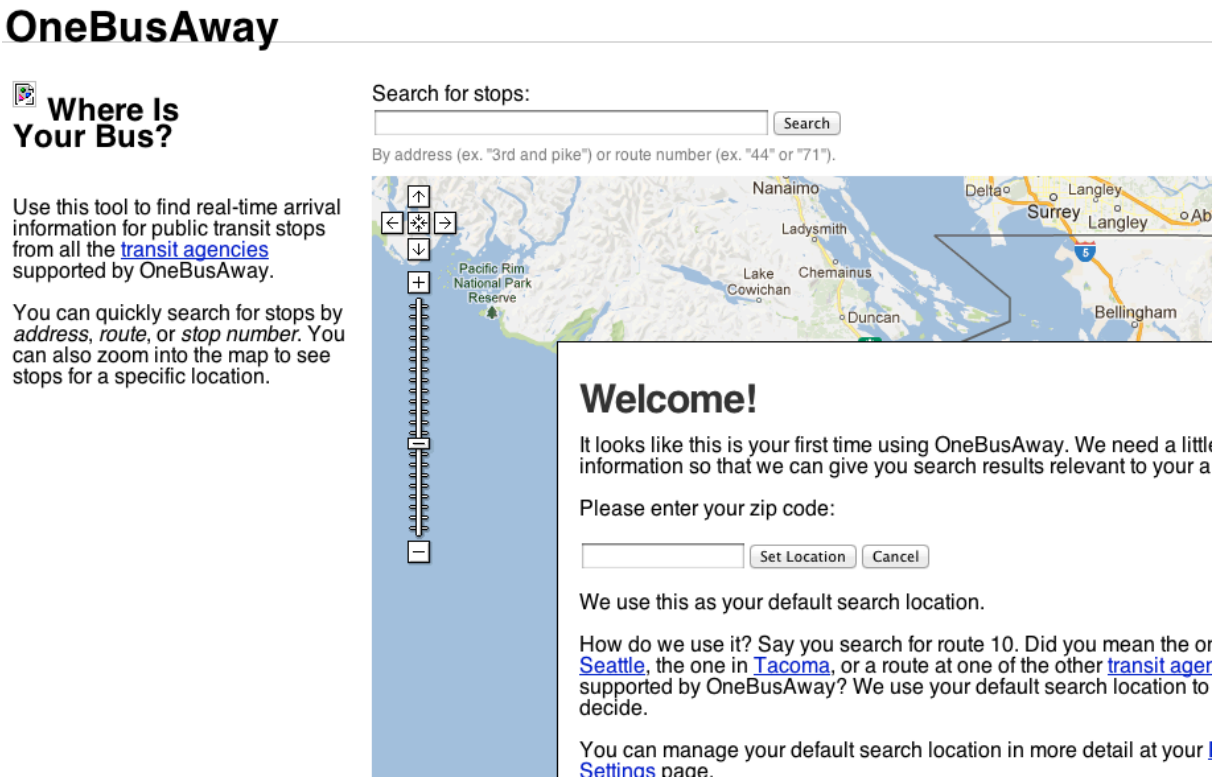
A aplicação OneBusAway Realtime Visualizer é uma simples aplicação que lê dados GTFS-realtime do tipo VehiclePosition e exibe em uma mapa a trajetória dos veículos.

Figura 40 - OneBusAway Realtime Visualizer



Já a aplicação OneBusAway Quickstart Bundle é um conjunto de aplicações que tem como objetivo a visualização de dados GTFS e GTFS-realtime para o usuário final, ou seja, é uma aplicação equivalente ao serviço Google Transit.

Figura 41 - OneBusAway Bundle



5.1.3 Cenário de Teste

Para que os testes possam ser realizados, o sistema precisa possuir dados de alguma agência de trânsito. Diante disso, serão utilizados dados da Massachutsetts Bay Transportation Authority (MBTA) [37]. Já que os dados são muito extensos, somente um subconjunto desses serão utilizados. Esse subconjunto foi inserido no banco de dados e será exibido no ANEXO B através do formato JSON que é um formato que permite fácil leitura.

Em geral, uma agência será inserida no sistema, com apenas uma rota. Essa rota terá uma viagem. O identificador da viagem escolhida é 18002619. Baseado nisso, todos os horários de partida/chegada dessa viagem serão importados para o banco de dados, bem como todas as paradas de ônibus descritas pela MTBA.

5.2 Testes por Caso de Uso

5.2.1 Teste Check-in / Autenticar Veículo / Publicar Posição Veículo

Para realizar esses testes, é requerido que um veículo seja criado no sistema, e que o mesmo possua, por consequência, um usuário e uma senha. Será utilizado a aplicação SERVIDOR e a aplicação CLIENTE MÓVEL.

Ao fazer o Check-in, espera-se que o CLIENTE MÓVEL transmita a posição do veículo, automaticamente, quando o sistema notar uma mudança em sua localização. Devido a utilização de um emulador Android para execução do CLIENTE MÓVEL, é necessário que as mudanças de localização do dispositivo sejam enviadas através de um comando para o dispositivo.

Pode-se conectar no emulador Android através do seguinte comando:

telnet localhost 5554

Para enviar uma nova localização ao dispositivo, pode-se utilizar o comando:

geo fix <longitude value> <latitude value>

Ao enviar o comando com valores de latitude e longitude, o SERVIDOR recebeu a mensagem e salvou no banco de dados. Abaixo está a representação dos dados salvos no banco das colunas importantes.

{"creation_date": "2012-11-03 21:18:17", "vehicle_lat": "42.28465166666667",

"vehicle_lon": "-71.06448833333334", “vehicle_fk”=”1”}

5.2.2 Teste Obter GTFS.zip

Nesse teste, basta um pedido HTTP GET para /api/feed/gtfs/1. Ou seja, todos os dados de teste da agência MTBA serão convertidos para o formato GTFS. Abaixo está a imagem representando o teste.

Figura 42 - Teste de Caso de Uso: Obter GTFS.zip

5.2.3 Teste Obter Feed VehiclePosition

Para esse teste, basta fazer um pedido HTTP para /api/feed/vehicle-position/1. Caso o pedido seja feito com um parâmetro DEBUG=TRUE, os dados serão impressos na resposta HTTP como texto, ou seja, de forma legível por humanos. Abaixo está um exemplo de uma resposta:

Figura 43 - Teste de Caso de Uso: Obter Feed VehiclePosition

```
header {
  gtfs_realtime_version: "1.0"
  incrementality: FULL_DATASET
  timestamp: 1351987211
}
entity {
  id: "1"
  vehicle {
    position {
      latitude: 42.284653
      longitude: -71.06449
    }
    timestamp: 1351987098000
    vehicle {
      id: "1"
      label: "Vehicle!"
      license_plate: "LICENSEPLATE"
    }
  }
}
```

5.3 Testes de Integração

Nesse subcapítulo, serão executados testes de integração. O primeiro a ser realizado é aquele que utiliza a aplicação externa OneBusAway Realtime Visualizer. Esse teste mostrará que o sistema é capaz de gerar os dados da mensagem VehiclePosition seguindo o formato GTFS-realtime de forma correta. Em seguida, será utilizado o OneBusAway Quickstart Bundle para demonstrar que os dados gerados pela aplicação SERVIDOR podem ser utilizados para auxiliar os usuários finais dos serviços de transporte público.

5.3.1 Teste utilizando OneBusAway Realtime Visualizer

Para executar esse teste, é necessário que a aplicação SERVIDOR esteja rodando, e a aplicação CLIENTE MÓVEL esteja sendo executada e seja feita uma simulação para que a mesma transmita a mensagem VehiclePosition.

Foram dadas as seguintes localizações para o dispositivo móvel que está rodando o CLIENTE MÓVEL:

geo fix -71.064489 42.284652

geo fix -71.064489 42.284652

geo fix -71.065353 42.282407

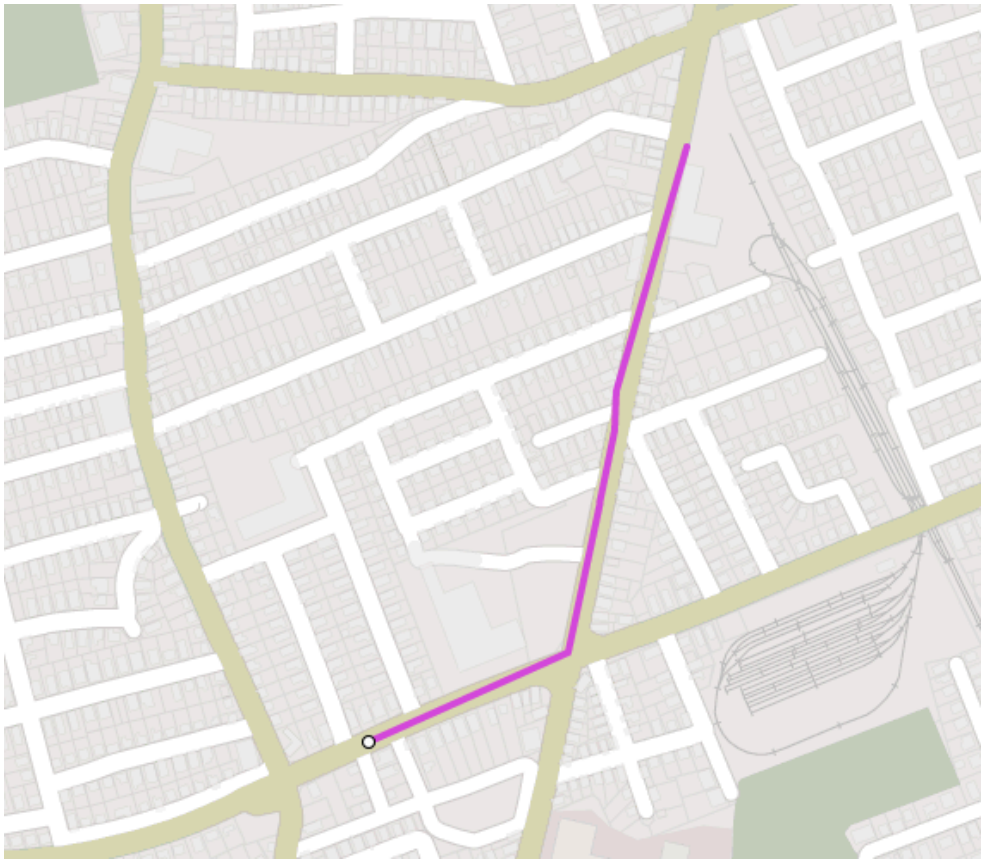
geo fix -71.065383 42.282032

geo fix -71.065963 42.279996

geo fix -71.068444 42.27917

Abaixo, pode-se observar que a aplicação Realtime Visualizer gerou a movimentação simulada do dispositivo.

Figura 44 - Teste do sistema utilizando OneBusAway Realtime Visualizer



5.3.2 Teste utilizando OneBusAway Quickstart Bundle

Nesse teste, o objetivo é utilizar a aplicação web do projeto OneBusAway para visualizar as chegadas e partidas previstas para uma determinada viagem salva no SERVIDOR. Como já apresentado anteriormente no ANEXO B, foi escolhido um subconjunto de dados da MTBA para execução dos testes.

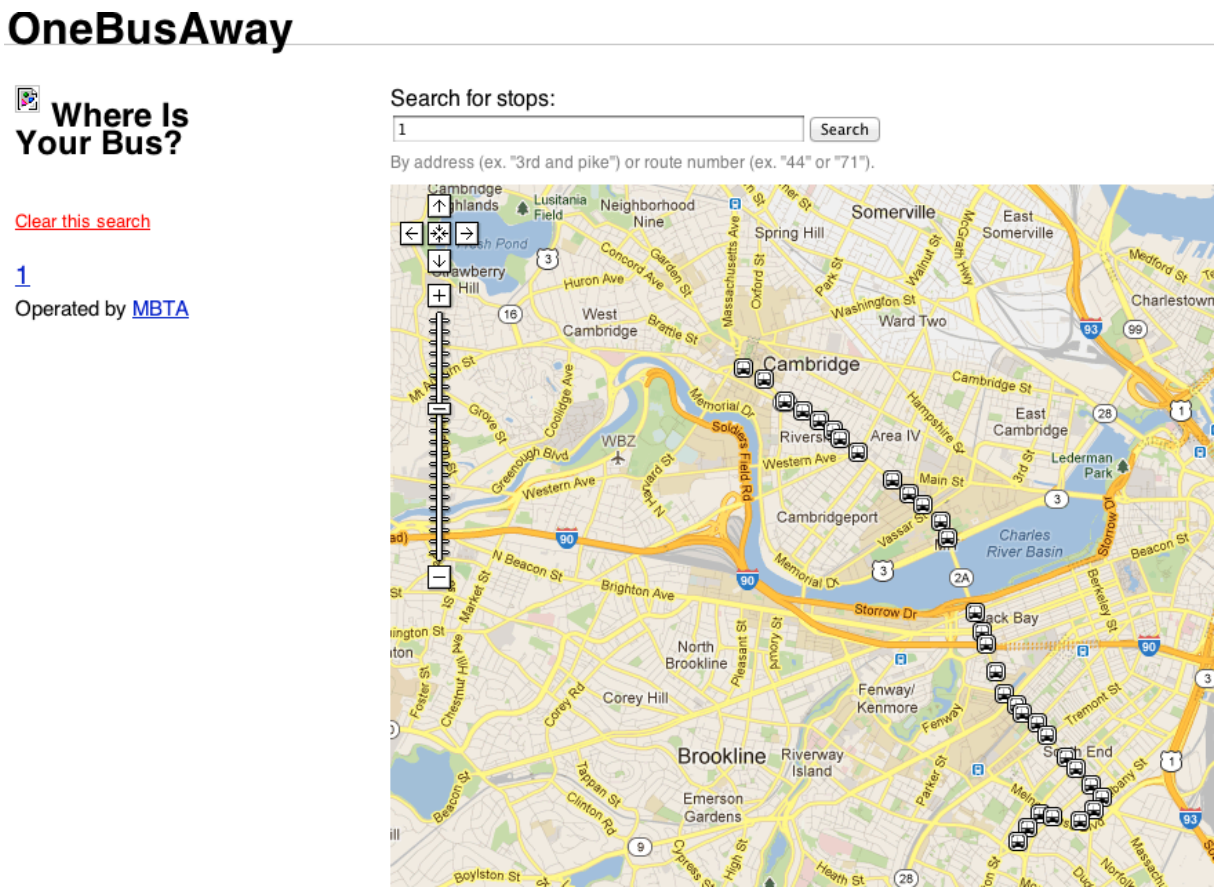
Para realização do teste, foi necessário a importação de dados GTFS em relação aos pontos de parada e aos horários de chegada/partida da viagem com identificador 18002619 da MTBA. Isso requereu a escrita de um programa em Java para leitura de arquivos GTFS e importação para o banco de dados.

Para execução da aplicacao OneBusAway é necessário a execução da mesma através do comando abaixo:

```
java -Xmx2G -Xms2G -XX:MaxPermSize=2048m  -server -jar
/Users/lucasschmidt/Desktop/onebusaway-quickstart-assembly-1.0.7-
webapp.war
```

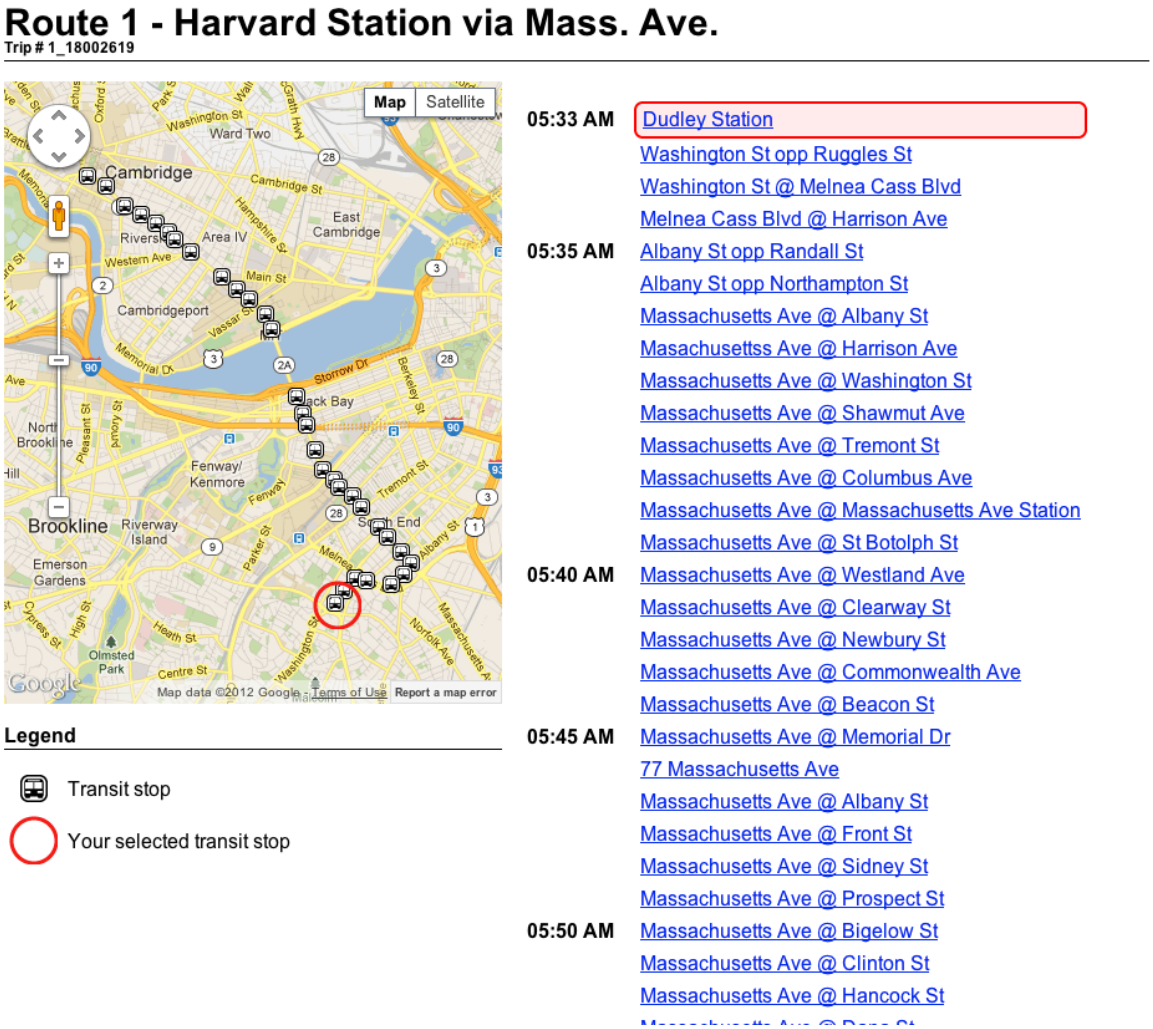
Procurando pela Rota 1, que é o nome da rota importada como cenário de teste, pode-se observar como o programa carregou os dados.

Figura 45 - Teste do sistema utilizando OneBusAway Bundle



Em seguida, pode-se observar, no ponto de vista da estação Dudley, todas as paradas que serão feitas por essa viagem.

Figura 46 - Teste do sistema utilizando OneBusAway Bundle



5.4 Conclusão dos Testes

Através dos testes dos principais Casos de Uso, conseguiu-se demonstrar que o sistema estava pronto para ser testado como um todo. Os testes de integração mostraram que o sistema está apto para comunicação através da especificação GTFS e GTFS-realtime.

5.5 Conclusões do trabalho

Esse trabalho foi realizado com o objetivo do desenvolvimento de uma ferramenta para auxiliar o monitoramento de dados em tempo real associados ao transporte público, especialmente aqueles dados que afetam diretamente os usuários.

Foi proposto um modelo de dados relacional capaz de representar dados necessários para a comunicação de um sistema, que armazena dados de transporte público, conforme as especificações GTFS e GTFS-realtime, com sistemas externos.

Foi desenvolvido uma aplicação, chamada de SERVIDOR, para armazenamento de dados conforme o modelo relacional proposto e para comunicação com sistemas externos seguindo as especificações GTFS e GTFS-realtime.

Foi desenvolvido uma aplicação, chamada de CLIENTE WEB, que permite a descrição de uma agência de trânsito conforme a especificação GTFS.

Foi desenvolvido uma aplicação, chamada de CLIENTE MÓVEL, que permite que veículos associados a agências possam transmitir dados em tempo real para o sistema SERVIDOR.

Testes foram feitos onde se mostrou que o sistema desenvolvido está apto para comunicação com outras aplicações através das especificações GTFS e GTFS-realtime.

Em geral, esse trabalho conseguiu explorar a especificação GTFS-realtime, ou seja, deu continuidade ao trabalho iniciado em JORGE [22], além de produzir entregáveis funcionais previstos nos objetivos desse trabalho.

Inobstante os aspectos supra referidos, ainda se deve considerar todo o conhecimento obtido durante o desenvolvimento do projeto pelo autor: somado a

conceitos de arquitetura de software orientada a serviços, conseguiu-se explorar o Spring Framework, a ferramenta Spring Roo, a biblioteca QueryDSL, a tecnologia GWT e desenvolvimento de aplicativos para o sistema operacional Android.

5.6 Sugestões de Trabalhos Futuros

Segue abaixo as sugestões de trabalhos futuros:

Desenvolvimento de sistemas capazes de extrair dados associados a qualidade do transporte público, ao trânsito de cidades, a otimização do número de veículos necessários para atender a sociedade, tomada de decisão por agências, além de outras informações não pensadas pelo autor.

Avaliação das aplicações do projeto OneBusAway para, tanto o melhoramento das mesmas, quanto até mesmo criação de novas aplicações capazes de exibir dados associados ao transporte público para a sociedade.

Análise de diversos dispositivos para detectar qual seria o mais indicado para uma implantação de um sistema capaz de comunicar dados em tempo real associados a um veículo de uma agência de trânsito.

REFERÊNCIAS

[1] KIZOOM, Nick; MILLE, Peter. **A Transmodel based XML schema for the Google Transit Feed Specification With a GTFS** / Transmodel comparison. Disponível em: <http://www.dft.gov.uk/transmodel/schema/doc/GoogleTransit/TransmodelForGoogle-09.pdf> Kizoom, 2008.

[2] ALEX, Ben. SCHMIDT, Stefan. STEWART, Alan. et al. Spring roo: reference documentation. Spring Framework. Spring Source website. Disponível em: <http://static.springsource.org/spring-roo/reference/html/>. Acesso em: 20 jun. 2012

[3] BELZ, Hans-Joachim. A class diagram of the gwt mvp model [update]. Disponível em:<http://minutefforts.com/blog/?p=50>. Acesso em: 19 jun. 2012.

[4] GOOGLE. **Especificação de feed do Google Transit**. Disponível em: <http://code.google.com/intl/pt-BR/transit/spec/transit_feed_specification.html>. Acesso em: 30 jun. 2009.

[5] GOOGLE I/O 2010: Architecting GWT apps. Google developers. Youtube. Disponível em: <http://www.youtube.com/watch?v=M5x6E6ze1x8>. Acesso em: 20 jun. 2012

[6] GOOGLE I/O 2010: Highly productive GWT. Google developers. Youtube. Disponível em: <http://www.youtube.com/watch?v=imiquTOLl64>. Acesso em: 20 jun. 2012

[7] GOOGLE I/O 2010: Highly productive GWT – best practices for writing smaller, faster apps. Google developers. Youtube. Disponível em: <http://www.youtube.com/watch?v=0F5zc1UAt2Y>. Acesso em: 20 jun. 2012

[8] GOOGLE DEVELOPERS. **GWT Development with Activities and Places**. Disponível em: <https://developers.google.com/web-toolkit/doc/latest/DevGuideMvpActivitiesAndPlaces>. Acesso em: 18 jun. 2012.

[9] GOOGLE DEVELOPERS. **Coding basis**: history. Disponível em: <https://developers.google.com/web-

toolkit/doc/latest/DevGuideCodingBasicsHistory> Acesso em: 18 jun. 2012.

[10] GOOGLE DEVELOPERS. **Getting Started with RequestFactory**. Disponível em: <https://developers.google.com/web-toolkit/doc/latest/DevGuideRequestFactory> Acesso em: 18 jun. 2012.

[11] GOOGLE DEVELOPERS. **Google web toolkit overview**. Disponível em: <https://developers.google.com/web-toolkit/overview> Acesso em: 19 jun. 2012

[12] GWT AND SPRING SECURITY. 19 may 2010. Disponível em: <http://technowobble.blogspot.com.br/2010/05/gwt-and-spring-security.html> Acesso em: 19 jun. 2012

[13] GWT Event Bus Basics. Jet brains. Vídeo online. Disponível em: <http://tv.jetbrains.net/videocontent/gwt-event-bus-basics>. Postado em: 25 feb. 2011

[14] HANSON, Robert. TACY, Adam. ESSINGTON, Jason. et al. **GWT in action**. 2. ed. Manning publications. 2012 Disponível em:< http://www.manning.com/tacy/GWTiA2_meap_ch01.pdf> Acesso em: 19 jun. 2012

[15] RAMSDALE, Chris. **Large scale application development and MVP**. Google developers. Updated mar. 2010. Disponível em: <https://developers.google.com/web-toolkit/articles/mvp-architecture>. Acesso em: 18 jun. 2012

[16] SPRING GWT software architecture. Disponível em: <http://crazygui.wordpress.com/2011/12/06/spring-gwt-software-architecture-for-scalable-applications-part-1> . Acesso em: 19 jun. 2012. 6 dez. 2011.

[17] WIKIPEDIA. **Service Oriented Architecture**. Disponível em: <http://en.wikipedia.org/wiki/Service_Oriented_Architecture> Acesso em: 19 jun. 2012

[18] DE, Piotrek. **GWT**: Authentication for some part of application using GWT login page. Stackoverflow website. Edited in: 28 jun. 2011. Disponível em: http://stackoverflow.com/questions/6508238/gwt-authentication-for-some-part-of-application-using-gwt-login-page. Acesso em: 18 jun. 2012

[19] GIERKE, Oliver. **Advanced Spring Data JPA – Specifications and QueryDSL**. Disponível em: http://blog.springsource.org/2011/04/26/advanced-spring-data-jpa-specifications-and-querydsl. Acesso em 15 jun. 2012

[20] Spring Social Showcase. Github website. Disponível em: https://github.com/SpringSource/spring-social-samples/tree/master/spring-social-showcase . Último acesso em 23 jun. 2012

[21] GWT and Spring Security. Disponível em: <http://technowobble.blogspot.com.br/2010/05/gwt-and-spring-security.html>. Último acesso em 23 jun. 2012

[22] JORGE, Júlio Cesar. **Desenvolvimento de uma aplicação para consulta de itinerários e horários do transporte público em dispositivo móvel baseado na arquitetura SOA**. Monografia (Graduação em Sistemas de Informação). UFSC: Florianópolis. 139p. 2011

[23] One Bus Away. Disponível em http://www.onebusaway.org. Último acesso em 23 jun 2012

[24] GTFS-Realtime Reference. Disponível em https://developers.google.com/transit/gtfs-realtime/reference. Último acesso em 29 jun 2012

[25] SchemaSpy. Disponível em http://schemaspy.sourceforge.net/. Último acesso em 20 out 2012.

[26] MVP4G. Disponível em http://code.google.com/p/mvp4g/. Último acesso em 21 out 2012.

[27] RestyGWT. Disponível em http://restygwt.fusesource.org/. Último acesso em 21 out 2012.

[28] MYSQL Workbench. Disponível em http://www.mysql.com/downloads/workbench/. Último acesso em 21 out 2012.

[29] Use Case. Disponível em http://en.wikipedia.org/wiki/Use_case#Use_case_structure. Último acesso em 27 out 2012.
[30] CRUD. Disponível em http://en.wikipedia.org/wiki/Create,_read,_update_and_delete. Último acesso em 27 out 2012.

[31] Hibernate Validator. Disponível em http://www.hibernate.org/subprojects/validator.html. Último acesso em 27 out 2012.

[32] WADL – Web Application Description Language. Disponível em http://en.wikipedia.org/wiki/Web_Application_Description_Language. Último acesso em 27 out 2012.

[33] Task Execution and Scheduling. Disponível em http://static.springsource.org/spring/docs/3.0.x/reference/scheduling.html. Último acesso em 27 out 2012.

[34] Biblioteca onebusaway-gtfs-realtime-exporter. Disponível em https://github.com/OneBusAway/onebusaway-gtfs-realtime-exporter/wiki. Último acesso em 27 out 2012.

[35] OneBusAway Realtime Data Visualizer. Disponível em https://github.com/OneBusAway/onebusaway-gtfs-realtime-visualizer/wiki. Último acesso em 3 nov 2012.

[36] OneBusAway Quickstart Bundle. Disponível em http://developer.onebusaway.org/modules/onebusaway-application-modules/current/guides/quickstart-guide.html. Último acesso em 3 nov 2012.

[37] MBTA Developers. Disponível em http://www.mbta.com/rider_tools/developers/. Último acesso em 3 nov 2012.

ANEXO A – MENSAGENS GTFS-REALTIME

Esse anexo representa as mensagens definidas pelo padrão GTFS-realtime. Retirado de [24].

GTFS-realtime Reference

A GTFS-realtime feed lets transit agencies provide consumers with realtime information about disruptions to their service (stations closed, lines not operating, important delays, etc.) location of their vehicles, and expected arrival times.

Version 1.0 of the feed specification is discussed and documented on this site.

Term Definitions

required: Exactly one

repeated: Zero or more

message: Complex type

enum: List of fixed values

Elements

message FeedMessage

The contents of a feed message. Each message in the stream is obtained as a response to an appropriate HTTP GET request. A real-time feed is always defined with relation to an existing GTFS feed. All the entity ids are resolved with respect to the GTFS feed.

A feed depends on some external configuration:

The corresponding GTFS feed.

Feed application (updates, positions or alerts). A feed should contain only items of the appropriate applications; all the other entities will be ignored.

Polling frequency, controlled by min_update_delay, max_update_delay.

Fields

Field Name	*Type*	*Cardinality*	*Description*
Header	FeedHeader	required	Metadata about this feed and feed message.

Entity	FeedEntity	repeated	Contents of the feed.

message FeedHeader

Metadata about a feed, included in feed messages.

Fields

Field Name	*Type*	*Cardinality*	*Description*
gtfs_realtime_version	string	required	Version of the feed specification. The current version is 1.0.
incrementality	Incrementality	optional	
Timestamp	uint64	optional	This timestamp identifies the moment when the content of this feed has been created (in server time). In POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC). To avoid time skew between systems producing and consuming realtime information it is strongly advised to derive timestamp from a time server. It is completely acceptable to use Stratum 3 or even lower strata servers since time differences up to a couple of seconds are tolerable.

enum Incrementality

Determines whether the current fetch is incremental.

FULL_DATASET: this feed update will overwrite all preceding realtime information for the feed. Thus this update is expected to provide a full snapshot of all known realtime information.

DIFFERENTIAL: currently, this mode is **unsupported** and behavior is **unspecified** for feeds that use this mode. There are discussions on the GTFS-realtime mailing-listaround fully specifying the behavior of DIFFERENTIAL mode and the documentation will be updated when those discussions are finalized.

Values

Value
FULL_DATASET

DIFFERENTIAL

message FeedEntity

A definition (or update) of an entity in the transit feed. If the entity is not being deleted, exactly one of 'trip_update', 'vehicle' and 'alert' fields should be populated.

Fields

Field Name	*Type*	*Cardinality*	*Description*
Id	string	Required	Feed-unique identifier for this entity. The ids are used only to provide incrementality support. The actual entities referenced by the feed must be specified by explicit selectors (see EntitySelector below for more info).
is_deleted	bool	Optional	Whether this entity is to be deleted. Relevant only for incremental fetches.
trip_update	TripUpdate	Optional	Data about the realtime departure delays of a trip.
Vehicle	VehiclePosition	Optional	Data about the realtime position of a vehicle.
Alert	Alert	Optional	Data about the realtime alert.

message TripUpdate

Real-time update of the progress of a vehicle along a trip. Depending on the value of ScheduleRelationship, a TripUpdate can specify:

A trip that proceeds along the schedule.

A trip that proceeds along a route but has no fixed schedule.

A trip that has been added or removed with regard to schedule.

The updates can be for future, predicted arrival/departure events, or for past events that already occurred. In most cases information about past events is a measured value thus its uncertainty value is recommended to be 0. Although there could be cases when this does not hold so it is allowed to have uncertainty value different from 0 for past events. If an update's uncertainty is not 0, either the update is an approximate prediction for a trip that has not completed or the measurement is not precise or the update was a prediction for the past that has not been verified after the event occurred.

Note that the update can describe a trip that has already completed. To this end, it is enough to provide an update for the last stop of the trip. If the time of arrival at the last stop is in the past, the client will conclude that the whole trip is in the past (it is possible, although inconsequential, to also provide updates for preceding stops). This option is most relevant for a trip that has completed ahead of schedule, but according to the schedule, the trip is still proceeding at the current time. Removing the updates for this trip could make the client assume that the trip is still proceeding. Note that the feed provider is allowed, but not required, to purge past updates - this is one case where this would be practically useful.

Fields

Field Name	*Type*	*Cardinality*	*Description*
Trip	TripDescriptor	required	The Trip that this message applies to. There can be at most one TripUpdate entity for each actual trip instance. If there is none, that means there is no prediction information available. It does *not* mean that the trip is progressing according to schedule.
Vehicle	VehicleDescriptor	optional	Additional information on the vehicle that is serving this trip.
stop_time_update	StopTimeUpdate	repeated	Updates to StopTimes for the trip (both future, i.e., predictions, and in some cases, past ones, i.e., those that already happened). The updates must be sorted by stop_sequence, and apply for all the following stops of the trip up to the next specified one.

message StopTimeEvent

Timing information for a single predicted event (either arrival or departure). Timing consists of delay and/or estimated time, and uncertainty.

delay should be used when the prediction is given relative to some existing schedule in GTFS.

time should be given whether there is a predicted schedule or not. If both time and delay are specified, time will take precedence (although normally, time, if given for a scheduled trip, should be equal to scheduled time in GTFS + delay).

Uncertainty applies equally to both time and delay. The uncertainty roughly specifies the expected error in true delay (but note, we don't yet define its precise statistical meaning). It's possible for the uncertainty to be 0, for example for trains that are driven under computer timing control.

Fields

Field Name	*Type*	*Cardinality*	*Description*
Delay	int32	optional	Delay (in seconds) can be positive (meaning that the vehicle is late) or negative (meaning that the vehicle is ahead of schedule). Delay of 0 means that the vehicle is exactly on time.
Time	int64	optional	Event as absolute time. In POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC).
uncertainty	int32	optional	If uncertainty is omitted, it is interpreted as unknown. To specify a completely certain prediction, set its uncertainty to 0.

message StopTimeUpdate

Realtime update for arrival and/or departure events for a given stop on a trip. Updates can be supplied for both past and future events. The producer is allowed, although not required, to drop past events.

The update is linked to a specific stop either through stop_sequence or stop_id, so one of these fields must necessarily be set. See the documentation in TripDescriptor for more information.

Fields

Field Name	*Type*	*Cardinality*	*Description*
stop_sequence	uint32	optional	Must be the same as in stop_times.txt in the corresponding GTFS feed.
stop_id	string	optional	Must be the same as in stops.txt in the corresponding GTFS feed.
Arrival	StopTimeEvent	optional	
Departure	StopTimeEvent	optional	
schedule_relationship	ScheduleRelationship	optional	

enum ScheduleRelationship

The relation between this StopTime and the static schedule.

Values

Value	*Comment*
SCHEDULED	The vehicle is proceeding in accordance with its static schedule of stops, although not necessarily according to the times of the schedule. At least one of arrival and departure must be provided. If the schedule for this stop contains both arrival and departure times then so must this update. An update with only an arrival, say, where the schedule has both, indicates that the trip is terminating early at this stop.
SKIPPED	The stop is skipped, i.e., the vehicle will not stop at this stop. Arrival and departure are optional.
NO_DATA	No data is given for this stop. It indicates that there is no realtime information available. When set NO_DATA is propagated through subsequent stops so this is the recommended way of specifying from which stop you do not have realtime information. When NO_DATA is set neither arrival nor departure should be supplied.

message VehiclePosition

Realtime positioning information for a given vehicle.

Fields

Field Name	*Type*	*Cardinality*	*Description*
Trip	TripDescriptor	optional	The Trip that this vehicle is serving. Can be empty or partial if the vehicle can not be identified with a given trip instance.
Vehicle	VehicleDescriptor	optional	Additional information on the vehicle that is serving this trip. Each entry should have a **unique** vehicle id.
Position	Position	optional	Current position of this vehicle.
current_stop_sequence	uint32	optional	The stop sequence index of the current stop. The meaning of current_stop_sequence (i.e.,

			the stop that it refers to) is determined by current_status. If current_status is missing IN_TRANSIT_TO is assumed.
stop_id	string	optional	Identifies the current stop. The value must be the same as in stops.txt in the corresponding GTFS feed.
current_status	VehicleStopStatus	optional	The exact status of the vehicle with respect to the current stop. Ignored if current_stop_sequence is missing.
Timestamp	uint64	optional	Moment at which the vehicle's position was measured. In POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC).
congestion_level	CongestionLevel	optional	

enum VehicleStopStatus

Values

Value	*Comment*
INCOMING_AT	The vehicle is just about to arrive at the stop (on a stop display, the vehicle symbol typically flashes).
STOPPED_AT	The vehicle is standing at the stop.
IN_TRANSIT_TO	The vehicle has departed the previous stop and is in transit.

enum CongestionLevel

Congestion level that is affecting this vehicle.

Values

Value
UNKNOWN_CONGESTION_LEVEL

RUNNING_SMOOTHLY
STOP_AND_GO
CONGESTION
SEVERE_CONGESTION

message Alert

An alert, indicating some sort of incident in the public transit network.

Fields

Field Name	*Type*	*Cardinality*	*Description*
active_period	TimeRange	repeated	Time when the alert should be shown to the user. If missing, the alert will be shown as long as it appears in the feed. If multiple ranges are given, the alert will be shown during all of them.
informed_entity	EntitySelector	repeated	Entities whose users we should notify of this alert.
Cause	Cause	optional	
Effect	Effect	optional	
url	TranslatedString	optional	The URL which provides additional information about the alert.
header_text	TranslatedString	optional	Header for the alert. This plain-text string will be highlighted, for example in boldface.
description_text	TranslatedString	optional	Description for the alert. This plain-text string will be formatted as the body of the alert (or shown on an explicit "expand" request by the user). The information in the description should add to the information of the header.

enum Cause

Cause of this alert.

Values

Value
UNKNOWN_CAUSE
OTHER_CAUSE
TECHNICAL_PROBLEM
STRIKE
DEMONSTRATION
ACCIDENT
HOLIDAY
WEATHER
MAINTENANCE
CONSTRUCTION
POLICE_ACTIVITY
MEDICAL_EMERGENCY

enum Effect

The effect of this problem on the affected entity.

Values

Value
NO_SERVICE
REDUCED_SERVICE
SIGNIFICANT_DELAYS
DETOUR
ADDITIONAL_SERVICE

MODIFIED_SERVICE
OTHER_EFFECT
UNKNOWN_EFFECT
STOP_MOVED

message TimeRange

A time interval. The interval is considered active at time `t` if `t` is greater than or equal to the start time and less than the end time.

Fields

Field Name	*Type*	*Cardinality*	*Description*
start	uint64	optional	Start time, in POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC). If missing, the interval starts at minus infinity.
end	uint64	optional	End time, in POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC). If missing, the interval ends at plus infinity.

message Position

A geographic position of a vehicle.

Fields

Field Name	*Type*	*Cardinality*	*Description*
Latitude	float	required	Degrees North, in the WGS-84 coordinate system.
Longitude	float	required	Degrees East, in the WGS-84 coordinate system.
Bearing	float	optional	Bearing, in degrees, clockwise from True North, i.e., 0 is North and 90 is East. This can be the compass bearing, or the direction towards the next stop or intermediate location. This should not be deduced from the sequence of previous positions, which clients can compute from previous data.

Odometer	double	optional	Odometer value, in meters.
Speed	float	optional	Momentary speed measured by the vehicle, in meters per second.

message TripDescriptor

A descriptor that identifies an instance of a GTFS trip, or all instances of a trip along a route. To specify a single trip instance, the trip_id (and if necessary, start_time) is set. If route_id is also set, then it should be same as one that the given trip corresponds to. To specify all the trips along a given route, only the route_id should be set. Note that if the trip_id is not known, then station sequence ids in TripUpdate are not sufficient, and stop_ids must be provided as well. In addition, absolute arrival/departure times must be provided.

Fields

Field Name	*Type*	*Cardinality*	*Description*
trip_id	string	optional	The trip_id from the GTFS feed that this selector refers to. For non frequency expanded trips, this field is enough to uniquely identify the trip. For frequency expanded, start_time and start_date might also be necessary.
route_id	string	optional	The route_id from the GTFS that this selector refers to.
start_time	string	optional	The scheduled start time of this trip instance. This field should be given only if the trip is frequency-expanded in the GTFS feed. The value must precisely correspond to start_time specified for the route in the GTFS feed plus some multiple of headway_secs. Format of the field is same as that of GTFS/frequencies.txt/start_time, e.g., 11:15:35 or 25:15:35.
start_date	string	optional	The scheduled start date of this trip instance. This field must be provided to disambiguate

			trips that are so late as to collide with a scheduled trip on a next day. For example, for a train that departs 8:00 and 20:00 every day, and is 12 hours late, there would be two distinct trips on the same time. This field can be provided but is not mandatory for schedules in which such collisions are impossible - for example, a service running on hourly schedule where a vehicle that is one hour late is not considered to be related to schedule anymore. In YYYYMMDD format.
schedule_relationship	ScheduleRelationship	optional	

enum ScheduleRelationship

The relation between this trip and the static schedule. If a trip is done in accordance with temporary schedule, not reflected in GTFS, then it shouldn't be marked as SCHEDULED, but marked as ADDED.

Values

Value	*Comment*
SCHEDULED	Trip that is running in accordance with its GTFS schedule, or is close enough to the scheduled trip to be associated with it.
ADDED	An extra trip that was added in addition to a running schedule, for example, to replace a broken vehicle or to respond to sudden passenger load.
UNSCHEDULED	A trip that is running with no schedule associated to it, for example, if there is no schedule at all.
CANCELED	A trip that existed in the schedule but was removed.
REPLACEMENT	A trip that replaces a portion of the static schedule. If the trip selector identifies a certain trip instance, then only that instance is replaced. If the selector identifies a route, then all the trips along that route are replaced.

	The replacement applies only to the portion of the trip supplied. For instance, consider a route that goes through stops A,B,C,D,E,F, and a REPLACEMENT trip provides data for stops A,B,C. Then, the times for stops D,E,F are still taken from the static schedule. A feed might supply several REPLACEMENT trips. In this case, the portion of the static schedule that is replaced is the union of what is defined by all the feeds. Normally, all the REPLACEMENT trips should either correspond to the same route or to individual trip instances.

message VehicleDescriptor

Identification information for the vehicle performing the trip.

Fields

Field Name	*Type*	*Cardinality*	*Description*
id	string	optional	Internal system identification of the vehicle. Should be **unique** per vehicle, and is used for tracking the vehicle as it proceeds through the system. This id should not be made visible to the end-user; for that purpose use the **label** field
label	string	optional	User visible label, i.e., something that must be shown to the passenger to help identify the correct vehicle.
license_plate	string	optional	The license plate of the vehicle.

message EntitySelector

A selector for an entity in a GTFS feed. The values of the fields should correspond to the appropriate fields in the GTFS feed. At least one specifier must be given. If several are given, then the matching has to apply to all the given specifiers.

Fields

Field Name	*Type*	*Cardinality*	*Description*
agency_id	string	optional	
route_id	string	optional	
route_type	int32	optional	

trip	TripDescriptor	optional	
stop_id	string	optional	

message TranslatedString

An internationalized message containing per-language versions of a snippet of text or a URL. One of the strings from a message will be picked up. The resolution proceeds as follows: If the UI language matches the language code of a translation, the first matching translation is picked. If a default UI language (e.g., English) matches the language code of a translation, the first matching translation is picked. If some translation has an unspecified language code, that translation is picked.

Fields

Field Name	*Type*	*Cardinality*	*Description*
translation	Translation	repeated	At least one translation must be provided.

message Translation

A localized string mapped to a language.

Fields

Field Name	*Type*	*Cardinality*	*Description*
text	string	required	A UTF-8 string containing the message.
language	string	optional	BCP-47 language code. Can be omitted if the language is unknown or if no internationalization is done at all for the feed. At most one translation is allowed to have an unspecified language tag.

Last updated March 6, 2012.

ANEXO B – DADOS PARA TESTES

Nesse anexo, serão mostrados os dados utilizados como base dos testes. Não serão anexados todos os dados, apenas os principais. Abaixo constam descritos os dados da agência MTBA. Nome e fuso-horário.

MBTA America/Sao_Paulo

Em seguida, pode-se observar os dados da viagem escolhida como dado de teste. Respectivamente o código da viagem, o nome da viagem e o nome da rota.

18002619 Harvard Station via Mass. Ave. 1

Os outros dados restantes são os pontos de chegada/partida e os horários de parada nesses pontos.

PARTIDA	CHEGADA	NOME PONTO DE PARADA
5H33M	5H33M	Dudley Station
5H33M	5H33M	Washington St opp Ruggles St
5H34M	5H34M	Washington St @ Melnea Cass Blvd
5H34M	5H34M	Melnea Cass Blvd @ Harrison Ave
5H35M	5H35M	Albany St opp Randall St
5H36M	5H36M	Albany St opp Northampton St
5H36M	5H36M	Massachusetts Ave @ Albany St
5H36M	5H36M	Masachusettss Ave @ Harrison Ave
5H37M	5H37M	Massachusetts Ave @ Washington St
5H37M	5H37M	Massachusetts Ave @ Shawmut Ave
5H38M	5H38M	Massachusetts Ave @ Tremont St
5H39M	5H39M	Massachusetts Ave @ Columbus Ave
5H39M	5H39M	Massachusetts Ave @ Massachusetts Ave Station
5H39M	5H39M	Massachusetts Ave @ St Botolph St
5H40M	5H40M	Massachusetts Ave @ Westland Ave
5H41M	5H41M	Massachusetts Ave @ Clearway St
5H42M	5H42M	Massachusetts Ave @ Newbury St
5H42M	5H42M	Massachusetts Ave @ Commonwealth Ave
5H43M	5H43M	Massachusetts Ave @ Beacon St
5H45M	5H45M	Massachusetts Ave @ Memorial Dr
5H45M	5H45M	77 Massachusetts Ave
5H46M	5H46M	Massachusetts Ave @ Albany St

5H47M 5H47M Massachusetts Ave @ Front St

5H47M 5H47M Massachusetts Ave @ Sidney St

5H49M 5H49M Massachusetts Ave @ Prospect St

5H50M 5H50M Massachusetts Ave @ Bigelow St

5H50M 5H50M Massachusetts Ave @ Clinton St

5H51M 5H51M Massachusetts Ave @ Hancock St

5H51M 5H51M Massachusetts Ave @ Dana St

5H52M 5H52M Massachusetts Ave @ Trowbridge St

5H53M 5H53M Massachusetts Ave @ Quincy St

5H53M 5H53M Massachusetts Ave @ Holyoke St - Holyoke Gate

Por fim, pode-se observar os dados criados para representação de um veículo dessa agência. Nome do veículo, placa, o usuário para autenticação e a senha do mesmo.

Vehicle! LICENSEPLATE vehicle test

ANEXO C - DESENVOLVIMENTO DE UMA APLICAÇÃO PARA MONITORAÇÃO DE INFORMAÇÕES EM TEMPO REAL, ASSOCIADAS AO TRANSPORTE PUBLICO

Lucas Thum Silveira Schmidt[1]

RESUMO

Esse artigo descreve a proposição e o desenvolvimento de uma aplicação para permitir a monitoração de dados em tempo real, associadas ao transporte público. Baseado na especificação GTFS-realtime (Google- Transit-Feed-Specification), associada ao Google Transit, é possível a utilização de uma aplicação para disseminação de dados em tempo real como posição de um ônibus em um determinado instante, mudanças na agenda em uma linha de ônibus ou até mesmo o estado de uma estação de ônibus.

Palavras-Chave: Transporte Público. GTFS. GTFS-realtime. Google Transit. Tempo Real. Serviços-web. Dispositivos Móveis.

ABSTRACT

This article describes the proposition and the development of an application to allow the monitoring of real-time data, related to public transportation systems. Based on GTFS-realtime (Google- Transit-Feed-Specification), associated to Google Transit, its possible to utilize an application to broadcast real-time data like the position of a bus in an instant, changes in the schedule of a bus route or even the state of a bus station.

[1] Graduando em Ciência da Computação - Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil – lucas.schmidt@inf.ufsc.br

Keywords: Public Transportation. GTFS. GTFS-realtime. Google Transit. Real-time. Web-services. Mobile devices.

1 INTRODUÇÃO

Nos próximos anos, o Brasil será sede de dois grandes eventos em 2014 e em 2016: a Copa do Mundo de Futebol e as Olimpíadas, respectivamente, e visitado por turistas de nacionalidades diversas. Dentre os grandes problemas de infraestrutura das cidades brasileiras, um dos principais certamente é o transporte público. Os turistas, bem como os moradores das grandes cidades necessitam de informações precisas acerca do meio de transporte a ser utilizado em seu trajeto e em que momento o mesmo estará disponível. Entretanto, esse tipo de serviço ainda não é oferecido na maioria das cidades.

Diante disso, as nossas cidades precisam de uma solução para viabilizar a comunicação e disponibilizar informações acerca dos serviços de transporte público a todas essas pessoas. O Google possui um serviço chamado de Google Transit, responsável por distribuir dados de rotas de transporte público e informações em tempo real, tais como a posição de um veículo e o estado de uma viagem programada. Através de um serviço desses é possível obter uma forma de centralizar todas as informações dos órgãos ou empresas responsáveis pelo transporte.

Mesmo existindo uma forma de centralização de todos os dados relacionados aos serviços de transporte público, resta, ainda, o desafio de como cada órgão ou empresa irá obter esses dados em tempo real. Além disso, como esses dados serão disponibilizados seguindo um modelo de dados de um serviço como o Google Transit.

1.1 Problema

Para disponibilizar dados de transporte público de forma centralizada ao público em geral, existem serviços como o Google Transit. Entretanto, para obter esses dados não há um procedimento simplificado, já que cada órgão ou empresa possui seu próprio sistema

computacional. Não obstante isso, dados em tempo real precisam ser obtidos, diretamente, de todos os veículos, gerando custos e investimento nessa área de atuação. Para organizações de pequeno porte pode tornar-se inviável, pelo fato de não possuírem recursos suficientes para um investimento vultoso, ou seja, qualquer solução a ser proposta precisa apresentar uma significativa redução de custos.

Em JORGE (2011) , abordou-se o primeiro problema. Através do desenvolvimento de um sistema de cadastro de todos os órgãos e empresas responsáveis pelo transporte público, seguindo o modelo de dados do Google Transit, foi mostrado uma maneira de centralizar todos os serviços de transporte oferecidos por uma cidade em uma base de dados.

Em relação ao outro problema de como obter dados em tempo real dos veículos, existem as seguintes questões: qual tecnologia utilizar para transmitir esses dados de um veículo até um servidor; e que tipo de dispositivo e sistema será utilizado no veículo. Além disso, é necessário tornar esses dados disponíveis na internet seguindo um modelo de dados padronizado como o já apresentado em JORGE (2011).

1.2 Objetivo

O objetivo geral do é a proposição e o desenvolvimento de uma aplicação que permita a monitoração de informações associadas ao transporte público e compatíveis com o sistema Google Transit, com enfoque nas informações em tempo real.

2 PROJETO LÓGICO

2.1 Representação de Dados

Baseando-se em JORGE (2011) e no estudo feito acerca dos modelos de representação de dados de trânsito, será utilizado a especificação GTFS, com a diferença dessa apresentar a extensão para dados em tempo real.

GTFS ou “Google Transit Data Feed Specification” é uma especificação para representação de dados de agências de transporte público. Essa especificação permite que uma agência informe ao público a respeito do calendário de viagens programadas. Mais detalhes podem ser observados em JORGE (2011) ou KIZOOM (2008).

Já o GTFS-realtime é uma extensão do GTFS e é uma especificação que permite que agências de transporte publiquem os dados em tempo real de suas frotas.

Em JORGE (2011) utilizou a especificação GTFS, ou seja, teve como objetivo a manipulação de dados estáticos relativos ao transporte público. Já, através do GTFS-realtime, ora proposto, o modelo de dados representados deve ser ampliado para suportar dados como a posição e a velocidade de um veículo, o nível de congestionamento, alertas e outros dados que serão descritos em seguida.

O GTFS-realtime suporta três tipos de mensagens: “TripUpdate”, “VehiclePosition” e “Alert”. “TripUpdate” confere o poder, à agência de transporte público, de notificar o público em geral sobre atrasos em viagens, se uma viagem está seguindo sua programação, ou o veículo responsável pela viagem, enquanto o “VehiclePosition”, permite que a posição e a velocidade de um veículo seja divulgada. Por fim, a “Alert” irá permitir que avisos internacionalizados sejam publicados quando imprevistos acontecerem.

2.2 Modelo de dados proposto

Em KIZOOM (2008), pode-se observar um modelo entidade-relacionamento para representação do GTFS. Até o momento, não foram encontradas sugestões de um modelo para o GTFS-realtime. Diante disso, a seguir, será descrito o modelo compatível com GTFS-realtime que será utilizado no sistema a ser proposto, considerando as três novas mensagens adicionadas ao GTFS na especificação GTFS-realtime e a utilização do modelo encontrado no KIZOOM (2008, p. 19) como base.

Para melhor compreensão do modelo proposto, é necessário observar os seguintes conceitos em relação ao GTFS: AGENCY, representa uma agência de transporte público, STOP, representa uma parada de um veículo, ROUTE, ou rota, representa um grupo de TRIPS, ou viagens, que representam uma sequencia de duas ou mais paradas de veículo, STOP TIMES , representa o momento em que um veículo irá parar em uma viagem, e CALENDAR, representa o tempo de funcionamento de um serviço. Demais conceitos presentes no GTFS como TRANSFER, ZONE, entre outros, serão ignorados, por não serem obrigatórios nessa representação de dados. Na figura 1 pode-se observar o modelo entidade relacionamento simplificado do GTFS.

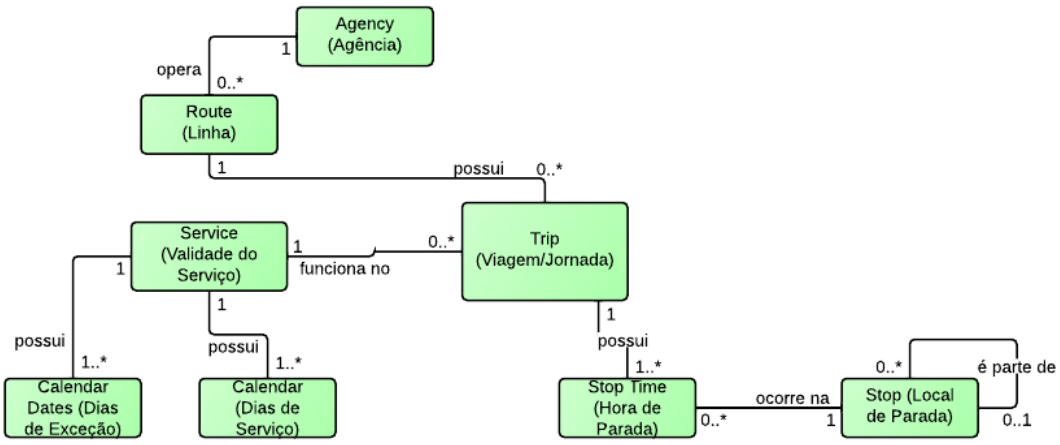


Figure 1 - Modelo Entidade-Relacionamento do GTFS

O modelo proposto é simplesmente uma extensão daquele apresentado em KIZOOM (2008, p. 19). Devido a adição de três novos tipos de mensagem no GTFS-realtime, novas entidades tiveram de ser criadas para representação dessas. Somado a isso, também foi necessário modelar os veículos. Na figura 2, pode-se observar o modelo entidade-relacionamento simplificado do GTFS-realtime.

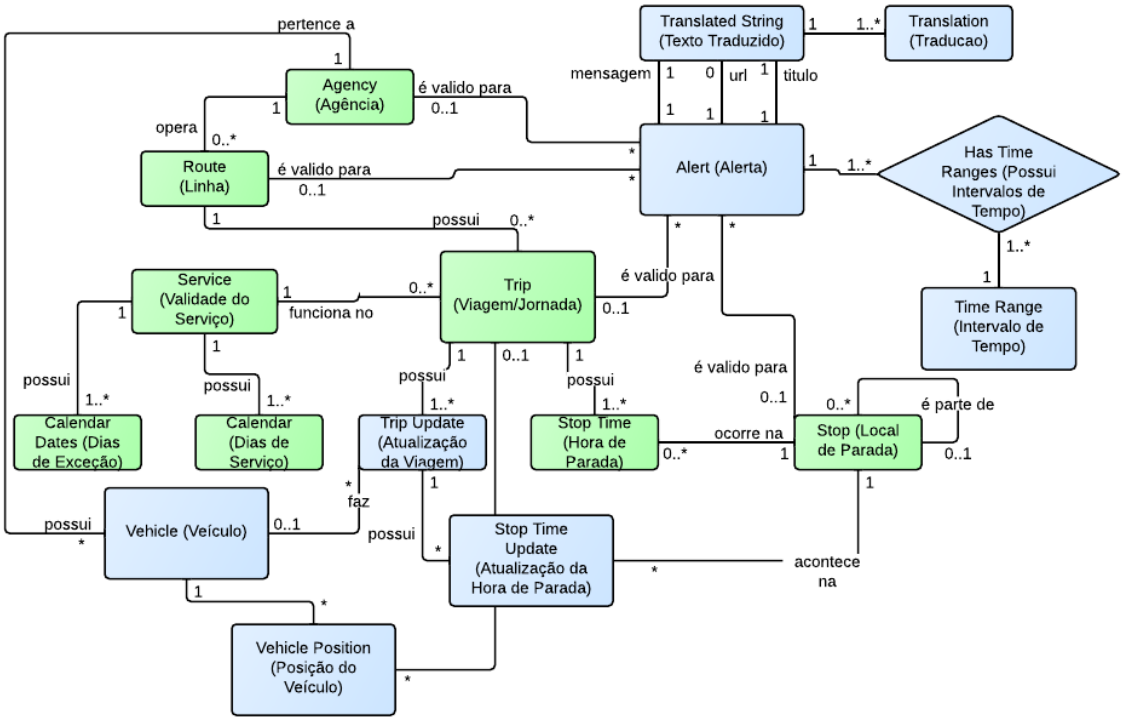


Figure 2 - Modelo Entidade-Relacionamento do GTFS-realtime

Pode-se observar que a entidade ALERT, que representa os alertas, especificados no GTFS-realtime, podem ser emitidos para diversos escopos: um alerta pode ser válido para uma agência, ou uma rota, ou uma parada de ônibus, representada pela entidade STOP, ou uma viagem, representada pela entidade TRIP. A entidade ALERT não foi normalizada devido a questões de performance: caso se conheça uma viagem, por exemplo, seria mais fácil fazer uma busca de quais alertas são válidos para essa, já que tudo poderia ser resolvido diretamente na cláusula “where” indicando a agência, a rota e a própria viagem como alternativas, sem a necessidade de junções.

Também pode-se observar no modelo entidade-relacionamento a entidade TRIP UPDATE, que representa a mensagem de atualização de uma viagem especificada no GTFS-realtime. Essa atualização é utilizada quando agências querem informar o público a respeito de atrasos de viagens, ou o veículo que está efetuando a viagem, ou caso a viagem tenha sido cancelada ou adicionada. Essa entidade deve ter relação com uma viagem, pode possuir uma

relação com um veículo, e diversas relações com a entidade STOP TIME UPDATE. Essa última representa uma atualização em relação a uma única parada durante uma viagem. Ou seja, significa o estado atual de uma parada de ônibus em relação a viagem. A parada nessa estação de ônibus pode ser cancelada, adicionada ou sofrer atrasos. Essa entidade representará parte de informação da mensagem de atualização de uma viagem especificada pelo GTFS-realtime. Terá relação com a entidade TRIP UPDATE e com a entidade STOP, que representa uma parada de ônibus.

Um veículo, representado pela entidade VEHICLE, deve possuir uma agência, e um usuário no sistema, representado pela entidade APP USER. Poderá possuir diversas atualizações de viagem, entidade TRIP UPDATE e atualizações de sua localização em um determinado instante, entidade VEHICLE POSITION.

Por fim, ainda em relação ao modelo entidade-relacionamento, na figura 2, podemos verificar a representação relacional da última mensagem especificada no GTFS-realtime: VEHICLE POSITION. Essa mensagem, representada por uma entidade de mesmo nome, informa ao público acerca da posição de um veículo em um determinado instante.

2.3 Sistema proposto

O sistema proposto pode ser dividido em dois grandes componentes: primeiramente, um servidor ,que é responsável por armazenar todos os dados em tempo real de veículos no formato GTFS-realtime e disponibilizar para o serviço Google Transit, e por fim, o cliente do servidor, que estará sendo executado em um dispositivo dentro de um veículo e terá função de transmitir os dados em tempo real para o servidor. Os dados que serão enviados ao servidor pela aplicação cliente são todos aqueles requeridos pelas mensagens do GTFS-realtime, mas, facilmente, podem ser ampliados, de acordo com as necessidades futuras.

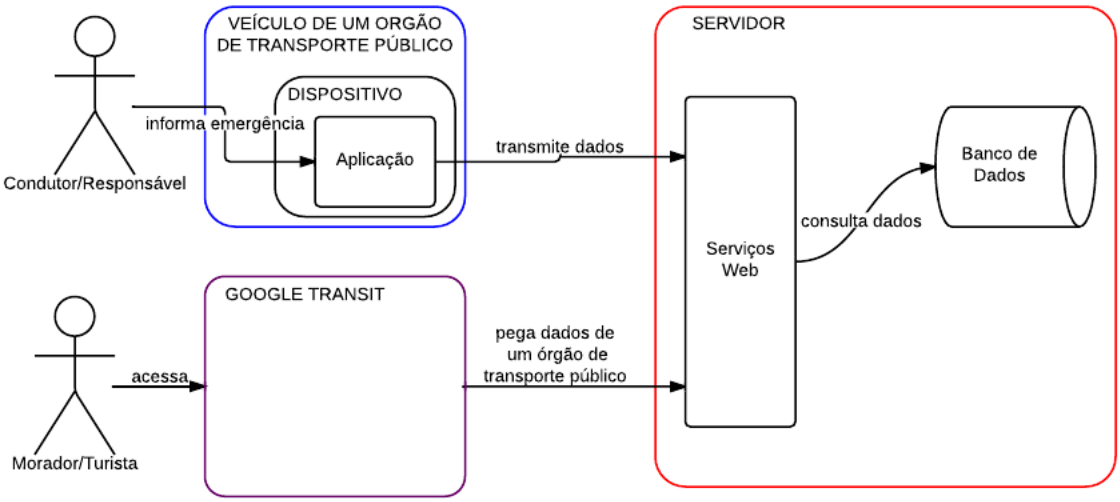


Figure 3 - Visão geral do sistema proposto

O SERVIDOR será responsável pelas seguintes funções:

- Armazenar dados das agências de trânsito e seus veículos;
- Disponibilizar dados para o Google Transit no formato GTFS-realtime;
- Receber dados relacionados as mensagens do GTFS-realtime de aplicações clientes instaladas em veículos;
- Precisará de um subcomponente chamado de SISTEMA PARA CADASTRO para permitir o cadastro de agências, rotas, veículos, e outros dados requeridos para modelar a estrutura de dados de um sistema de transporte público no formato GTFS;

Já o cliente no veículo, chamado de CLIENTE MÓVEL, terá a seguinte função:

- Transmitir dados requeridos pelo GTFS-realtime como a geo-localização de um veículo e alertas.

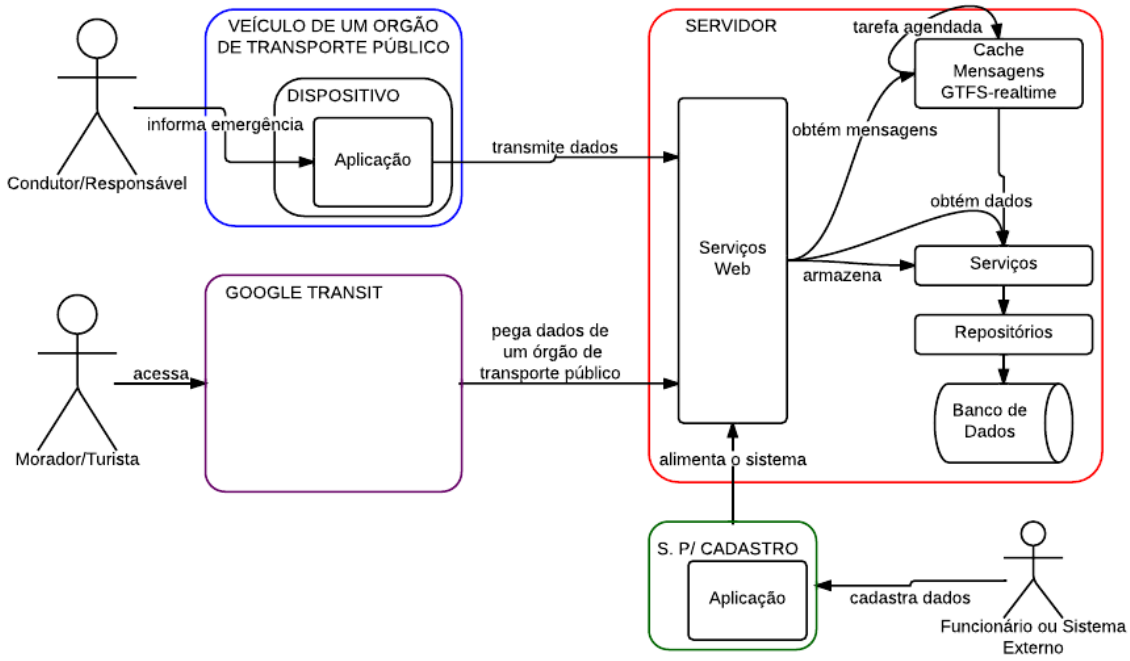


Figure 4 - Visão geral do sistema proposto ampliada

Na figura 4, é representado o SISTEMA PARA CADASTRO, que é um sistema externo responsável por alimentar o SERVIDOR com dados GTFS. Na prática, esse sistema poderia ser automatizado, ou seja, um sistema externo, de uma agência, por exemplo, poderia enviar os dados através de serviços web.

2.4 Requisitos do Sistema

Abaixo estão definidos os requisitos funcionais do SERVIDOR:

REF1. Receber a geo-localização de um veículo.

REF2. Receber alertas de um veículo.

REF3. Disponibilizar arquivos no formato GTFS e GTFS-realtime acessíveis através de serviços web para todo e qualquer usuário na internet.

REF4. Disponibilizar as operações de cadastrar e listar entidades que representam agências, viagens, veículos, rotas, paradas de veículos e disponibilidade de serviços através de serviços web protegidos, ou seja, restritos a usuários conhecidos pelo sistema.

A seguir estão definidos os requisitos não-funcionais do servidor:

RNF1. Servidor deverá ser feito em Java, utilizando o framework Spring Framework.

RNF2. Servidor deverá utilizar o MySQL como tecnologia do banco de dados relacional.

RNF3. Servidor deve utilizar REST como tecnologia dos serviços web para comunicação com o CLIENTE MÓVEL.

Seguem, os requisitos funcionais do SISTEMA PARA CADASTRO:

REF1. O sistema deve permitir a criação e a listagem de agências.

REF2. O sistema deve permitir a criação e a listagem de veículos.

REF3. O sistema deve permitir a criação e a listagem de viagens.

REF4. O sistema deve permitir a criação e a listagem de rotas.

REF5. O sistema deve permitir a criação e a listagem de paradas de veículos.

REF6. O sistema deve permitir a criação e a listagem de disponibilidades de serviço.

REF7. O sistema deve permitir a criação de usuários.

REF8. O sistema deve permitir a autenticação de um usuário cadastrado no mesmo.

RNF1. O sistema deve ser feito com a tecnologia GWT.

Abaixo estão definidos os requisitos funcionais da aplicação CLIENTE MÓVEL:

REF3. O sistema deve permitir a autenticação de um usuário do tipo veículo cadastrado no SISTEMA PARA CADASTRO.

REF2. O sistema deve enviar a geo-localização do veículo a cada 10 segundos.

REF3. O sistema deve permitir que o usuário escreva e envie alertas e escolha o tipo de alerta e o efeito desse conforme definido pelo GTFS-realtime na mensagem “Alert”.

Abaixo estão definidos os requisitos não-funcionais da aplicação cliente móvel:

RNF1. O sistema deve ser feito para o sistema operacional Android.

2.5 Métodos e Materiais

O SERVIDOR possuirá uma arquitetura SOA, ou SERVICE-ORIENTED-ARCHITECTURE, e possuirá serviços-web seguindo os conceitos REST.

Através da figura 5, é possível perceber todas as tecnologias que serão utilizadas para implantação desse sistema. Para o desenvolvimento do SERVIDOR que será responsável por armazenar os dados a respeito das empresas de trânsito seguindo o modelo GTFS-realtime, as tecnologias descritas abaixo serão utilizadas.

O Spring Roo será utilizado para desenvolvimento inicial da aplicação, ou seja, para a configuração inicial e geração de código. Será removido, a fim de permitir mais liberdade no desenvolvimento.

O Spring Framework será utilizado para o desenvolvimento da camada de serviço da aplicação, para controlar o acesso ao banco de dados, e para questões relacionadas à segurança, como restrição de acessos a determinados recursos de aplicação. Juntamente com esse, será utilizado o QueryDSL para definição das consultas ao banco de dados. Para gerenciar os serviços web será utilizado o Spring MVC, que é parte do Spring Framework.

Para conversão dos dados armazenados no banco de dados para o padrão GTFS-realtime, serão utilizados módulos do projeto One Bus Away [23], que é um projeto que possui como objetivo melhorar a usabilidade do transporte público através do desenvolvimento de aplicações para diversos dispositivos que informam dados de transporte público. Além disso, eles provem diversas bibliotecas e aplicativos open-source para auxiliar tanto usuários, como desenvolvedores e agências de trânsito.

O CLIENTE MÓVEL, que funcionará em veículos, será desenvolvido para o sistema operacional Android, ou seja, será necessário o uso de um aparelho. Essa escolha foi feita devido aos grandes recursos disponíveis em dispositivos Android como, por exemplo, o GPS e o acesso a internet, bem como a facilidade para testar o sistema.

Para desenvolvimento do SISTEMA PARA CADASTRO será utilizado o GWT e os frameworks MVP4G, que permite que a arquitetura Model View Presenter seja facilmente implantada, e RestyGWT, que permite fácil acesso a serviços-web.

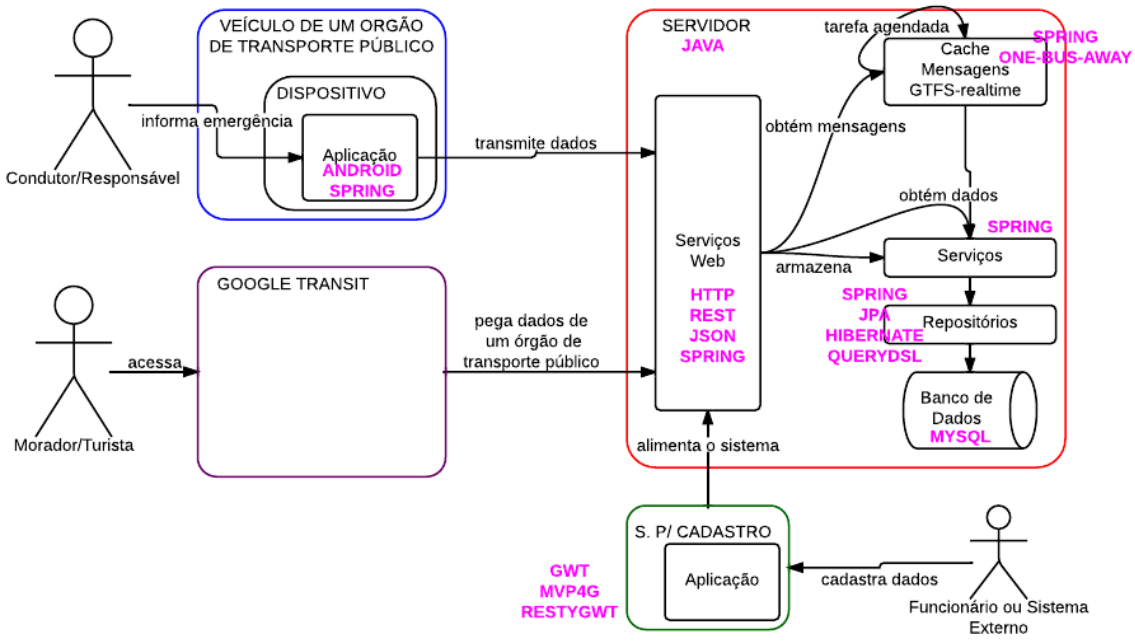


Figure 5 - Visão geral do sistema com Métodos e Materiais

3 RESULTADO DO DESENVOLVIMENTO

O desenvolvimento do SERVIDOR, que se iniciou a partir da geração inicial da camada de serviço pelo Spring Roo, foi um sucesso, já que o Spring Framework oferece todas as ferramentas necessárias para acesso a banco, gerenciamento do ciclo de vida dos objetos da aplicação e toda a segurança da mesma.

Em relação ao CLIENTE MÓVEL, o Android possui uma API que facilita muito o acesso a informações como a localização do aparelho. Isso permitiu o desenvolvimento rápido de uma aplicação que informa toda mudança de posição do aparelho. Obviamente, talvez

essa opção não seja a ideal em termos de custo/benefício, mas considerando que esse trabalho não tem enfoque em hardware, essa aplicação será muito útil para testes iniciais do projeto.

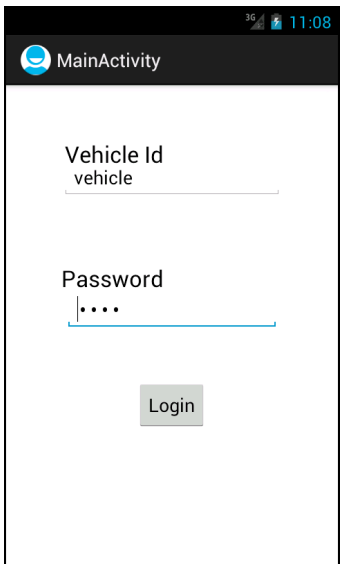


Figure 6 - CLIENTE MÓVEL

Já em relação ao SISTEMA PARA CADASTRO, caso o código desenvolvido para a aplicação seja comparado ao código que é gerado pelo Spring Roo para aplicações GWT, pode-se observar uma diferença muito grande na quantidade de códigos. O Spring Roo gera código a partir do framework padrão do GWT 2.4, o que dificulta muito, tanto o entendimento, como a customização do código. Por isso, foi decidido a utilização do framework MVP4G. O framework MVP4G proporcionou um desenvolvimento mais rápido e fácil do que a customização de um código gerado automático pelo Spring Roo. Já a biblioteca RestyGWT produz clientes de recursos REST de forma satisfatória, já que se pode acessar aos recursos REST do SERVIDOR sem muitos problemas, em um curto tempo, sem a necessidade de codificação de clientes.

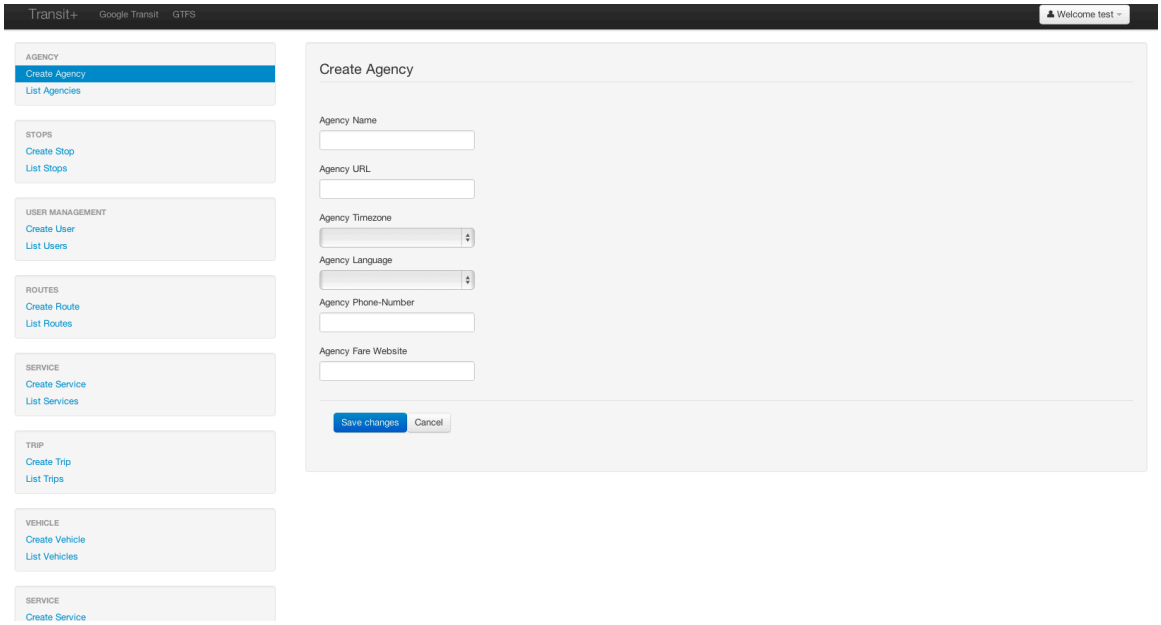


Figure 7 - Sistema para Cadastro

4 TESTES

4.1 Métodos e Materiais

Para que seja possível testar o sistema desenvolvido, é necessário a criação de um cenário em que uma agência de transporte público esteja cadastrada no sistema com todos os dados requeridos. Isso permitirá que os principais Casos de Uso possam ser testados. Além desses, testes de integração serão feitos.

Para que o sistema como um todo possa ser testado, será necessário a utilização de aplicações externas que possuam a capacidade de ler os dados no formato do GTFS e GTFS-realtime. Por isso, serão utilizadas as seguintes aplicações desenvolvidas como parte do projeto OneBusAway [23]: Realtime Visualizer [35] e Quickstart Bundle [36].

A aplicação OneBusAway Realtime Visualizer é uma simples aplicação que lê dados GTFS-realtime do tipo VehiclePosition e exibe em um mapa a trajetória dos veículos. Já a aplicação OneBusAway Quickstart Bundle é um conjunto de aplicações que tem como

objetivo a visualização de dados GTFS e GTFS-realtime para o usuário final, ou seja, é uma aplicação equivalente ao serviço Google Transit.

Figure 8 - OneBusAway Realtime Visualizer

Para que os testes possam ser realizados, o sistema precisa possuir dados de alguma agência de trânsito. Diante disso, serão utilizados dados da Massachutsetts Bay Transportation Authority (MBTA) [37]. Já que os dados são muito extensos, somente um subconjunto desses serão utilizados. Esse subconjunto foi inserido no banco de dados.

Em geral, uma agência será inserida no sistema, com apenas uma rota. Essa rota terá uma viagem. O identificador da viagem escolhida é 18002619. Baseado nisso, todos os horários de partida/chegada dessa viagem serão importados para o banco de dados, bem como todas as paradas de ônibus descritas pela MTBA.

4.2 Teste utilizando OneBusAway Realtime Visualizer

Para executar esse teste, é necessário que a aplicação SERVIDOR esteja rodando, e a aplicação CLIENTE MÓVEL esteja sendo executada e seja feita uma simulação para que a mesma transmita a mensagem VEHICLE POSITION.

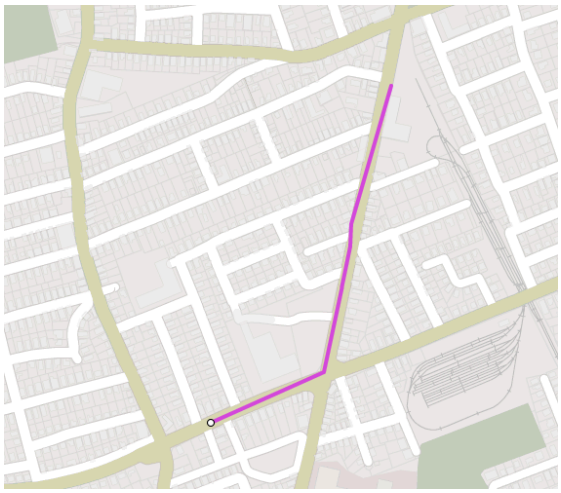


Figure 9 - Teste com OneBusAway Realtime Visualizer

Na figura 9, pode-se observar a imagem da aplicação OneBusAway Realtime Visualizer, após cinco comandos de geo-posicao foram enviados ao emulador Android, demonstrando a posição atual de um veículo simulado e toda a trajetória do mesmo.

4.3 Teste utilizando OneBusAway Quickstart Bundle

Nesse teste, o objetivo é utilizar a aplicação web do projeto OneBusAway para visualizar as chegadas e partidas previstas para uma determinada viagem salva no SERVIDOR. Como já dito anteriomente, foi escolhido um subconjunto de dados da MTBA para execução dos testes.

Para realização do teste, foi necessário a importação de dados GTFS em relação aos pontos de parada e aos horários de chegada/partida da viagem com identificador 18002619 da MTBA. Isso requereu a escrita de um programa em Java para leitura de arquivos GTFS e importação para o banco de dados.

Ao procurar pela Rota 1, que é o nome da rota importada como cenário de teste, pode-se verificar como o programa carregou os dados.

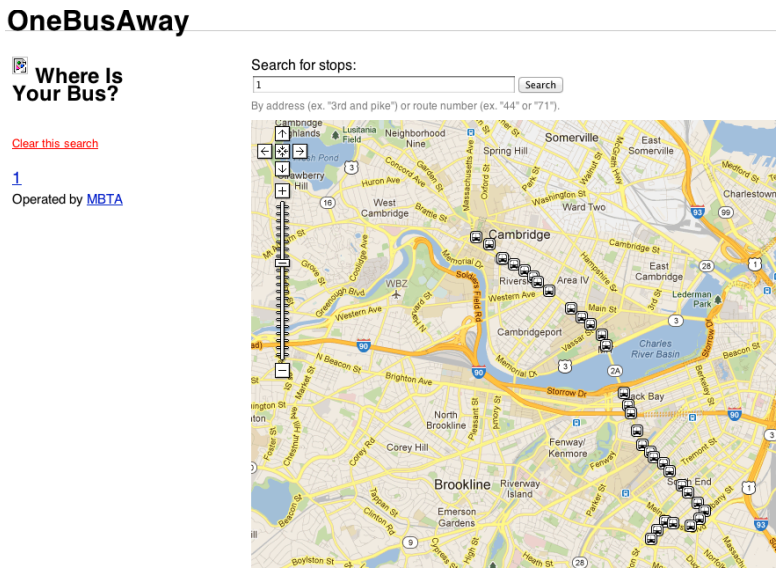


Figure 10 - Teste com OneBusAway Bundle

Em seguida, pode-se observar, no ponto de vista da estação Dudley, todas as paradas que serão feitas nessa viagem.

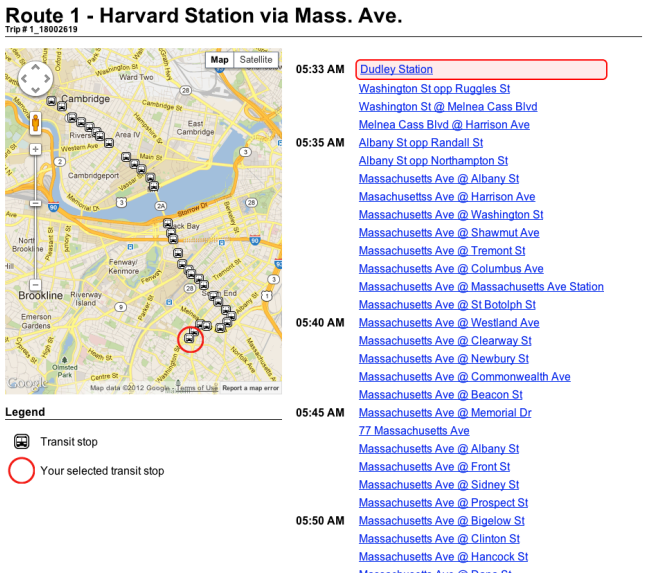


Figure 11 - Teste com OneBusAway Bundle

5 CONCLUSÕES

Esse trabalho foi realizado com o objetivo do desenvolvimento de uma ferramenta para auxiliar o monitoramento de dados, em tempo real, associados ao transporte público, especialmente aqueles dados que afetam diretamente aos usuários.

Foi proposto um modelo de dados relacional capaz de representar dados necessários para a comunicação de um sistema, que armazena dados de transporte público, conforme as especificações GTFS e GTFS-realtime, com sistemas externos.

Foi desenvolvido uma aplicação, chamada de SERVIDOR, para armazenamento de dados conforme o modelo relacional proposto e para comunicação com sistemas externos seguindo as especificações GTFS e GTFS-realtime.

Foi desenvolvido uma aplicação, chamada de SISTEMA PARA CADASTRO, que permite a descrição de uma agência de trânsito conforme a especificação GTFS.

Foi desenvolvido uma aplicação, chamada de CLIENTE MÓVEL, que permite que veículos associados a agências possam transmitir dados em tempo real para o sistema SERVIDOR.

Testes foram feitos onde se mostrou que o sistema desenvolvido está apto para comunicação com outras aplicações através das especificações GTFS e GTFS-realtime.

Em geral, esse trabalho conseguiu explorar a especificação GTFS-realtime, ou seja, deu continuidade ao trabalho iniciado em JORGE [22], além de produzir entregáveis funcionais previstos nos objetivos desse trabalho.

Inobstante os aspectos supra referidos, ainda se deve considerar todo o conhecimento obtido durante o desenvolvimento do projeto pelo autor: somado a conceitos de arquitetura de software orientada a serviços, conseguiu-se explorar o Spring Framework, a ferramenta Spring Roo, a biblioteca QueryDSL, a tecnologia GWT e desenvolvimento de aplicativos para o sistema operacional Android.

6 SUGESTÕES DE TRABALHOS FUTUROS

Seguem abaixo as sugestões de trabalhos futuros:

Desenvolvimento de sistemas capazes de extrair dados associados a qualidade do transporte público, ao trânsito de cidades, a otimização do número de veículos necessários

para atender a sociedade, tomada de decisão por agências, além de outras informações não pensadas pelo autor.

Avaliação das aplicações do projeto OneBusAway para, tanto o melhoramento das mesmas, quanto até mesmo criação de novas aplicações, capazes de exibir dados associados ao transporte público para a sociedade.

Análise de diversos dispositivos para detectar o mais indicado para uma implantação de um sistema capaz de comunicar dados, em tempo real, associados a um veículo de uma agência de trânsito.

Estudo em relação a alternativas de tecnologias para armazenamento de dados, como, por exemplo, banco de dados em grafos.

7 REFERÊNCIAS

ALEX, Ben. SCHMIDT, Stefan. STEWART, Alan. et al. Spring roo: reference documentation. Spring Framework. Spring Source website. Disponível em: <http://static.springsource.org/spring-roo/reference/html/>. Acesso em: 20 jun. 2012

BELZ, Hans-Joachim. A class diagram of the gwt mvp model [update]. Disponível em:<http://minutefforts.com/blog/?p=50>. Acesso em: 19 jun. 2012.

Biblioteca onebusaway-gtfs-realtime-exporter. Disponível em https://github.com/OneBusAway/onebusaway-gtfs-realtime-exporter/wiki. Último acesso em 27 out 2012.

CRUD. Disponível em http://en.wikipedia.org/wiki/Create,_read,_update_and_delete. Último acesso em 27 out 2012.

DE, Piotrek. GWT: Authentication for some part of application using GWT login page. Stackoverflow website. Edited in: 28 jun. 2011. Disponível em: http://stackoverflow.com/questions/6508238/gwt-authentication-for-some-part-of-application-using-gwt-login-page. Acesso em: 18 jun. 2012

GIERKE, Oliver. **Advanced Spring Data JPA**: Specifications and QueryDSL. Disponível em: http://blog.springsource.org/2011/04/26/advanced-spring-data-jpa-specifications-and-querydsl. Acesso em 15 jun. 2012

GOOGLE. **Especificação de feed do Google Transit**. Disponível em: <http://code.google.com/intl/pt- BR/transit/spec/transit_feed_specification.html>. Acesso em: 30 jun. 2009.

GOOGLE I/O 2010**: Architecting GWT apps**. Google developers. Youtube. Disponível em: <http://www.youtube.com/watch?v=M5x6E6ze1x8>. Acesso em: 20 jun. 2012

GOOGLE I/O 2010: **Highly productive GWT**. Google developers. Youtube. Disponível em: <http://www.youtube.com/watch?v=imiquTOLl64>. Acesso em: 20 jun. 2012

GOOGLE I/O 2010: **Highly productive** GWT: best practices for writing smaller, faster apps. Google developers. Youtube. Disponível em: <http://www.youtube.com/watch?v=0F5zc1UAt2Y>. Acesso em: 20 jun. 2012

GOOGLE DEVELOPERS. **GWT Development with Activities and Places**. Disponível em: <https://developers.google.com/web-toolkit/doc/latest/DevGuideMvpActivitiesAndPlaces>. Acesso em: 18 jun. 2012.

GOOGLE DEVELOPERS. **Coding basis**: history. Disponível em: <https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsHistory> Acesso em: 18 jun. 2012.

GOOGLE DEVELOPERS. **Getting Started with RequestFactory**. Disponível em: <https://developers.google.com/web-toolkit/doc/latest/DevGuideRequestFactory> Acesso em: 18 jun. 2012.

GOOGLE DEVELOPERS. **Google web toolkit overview**. Disponível em: <https://developers.google.com/web-toolkit/overview> Acesso em: 19 jun. 2012

GTFS-Realtime Reference. Disponível em <https://developers.google.com/transit/gtfs-realtime/reference>. Último acesso em 29 jun. 2012

GWT AND SPRING SECURITY. 19 may 2010. Disponível em: <http://technowobble.blogspot.com.br/2010/05/gwt-and-spring-security.html> Acesso em: 19 jun. 2012

GWT AND SPRING SECURITY. Disponível em: <http://technowobble.blogspot.com.br/2010/05/gwt-and-spring-security.html>. Último acesso em 23 jun. 2012

GWT Event Bus Basics. **Jet brains**. Vídeo online. Disponível em: <http://tv.jetbrains.net/videocontent/gwt-event-bus-basics>. Postado em: 25 feb. 2011

HANSON, Robert. TACY, Adam. ESSINGTON, Jason. et al. **GWT in action**. 2. ed. Manning publications. 2012 Disponível em:< http://www.manning.com/tacy/GWTiA2_meap_ch01.pdf> Acesso em: 19 jun. 2012

Hibernate Validator. Disponível em http://www.hibernate.org/subprojects/validator.html. Último acesso em 27 out 2012.

JORGE, Júlio Cesar. **Desenvolvimento de uma aplicação para consulta de itinerários e horários do transporte público em dispositivo móvel baseado na arquitetura SOA**. Monografia (Graduação em Sistemas de Informação). UFSC: Florianópolis. 139p. 2011

KIZOOM, Nick; MILLE, Peter. A Transmodel based XML schema for the Google Transit Feed Specification With a GTFS / Transmodel comparison. Disponível em: <http://www.dft.gov.uk/transmodel/schema/doc/GoogleTransit/TransmodelForGoogle-09.pdf> Kizoom, 2008.

MBTA Developers. Disponível em http://www.mbta.com/rider_tools/developers/. Último acesso em 3 nov 2012.

MVP4G. Disponível em http://code.google.com/p/mvp4g/. Último acesso em 21 out 2012.

MYSQL Workbench. Disponível em http://www.mysql.com/downloads/workbench/. Último acesso em 21 out 2012.

One Bus Away. Disponível em http://www.onebusaway.org. Último acesso em 23 jun 2012

OneBusAway Quickstart Bundle. Disponível em http://developer.onebusaway.org/modules/onebusaway-application-modules/current/guides/quickstart-guide.html. Último acesso em 3 nov 2012.

OneBusAway Realtime Data Visualizer. Disponível em https://github.com/OneBusAway/onebusaway-gtfs-realtime-visualizer/wiki. Último acesso em 3 nov 2012.

RAMSDALE, Chris. **Large scale application development and MVP**. Google developers. Updated mar. 2010. Disponível em: <https://developers.google.com/web-toolkit/articles/mvp-architecture>. Acesso em: 18 jun. 2012

RestyGWT. Disponível em http://restygwt.fusesource.org/. Último acesso em 21 out 2012.

SchemaSpy. Disponível em http://schemaspy.sourceforge.net/. Último acesso em 20 out 2012.

SPRING GWT software architecture. Disponível em: <http://crazygui.wordpress.com/2011/12/06/spring-gwt-software-architecture-for-scalable-applications-part-1> . Acesso em: 19 jun. 2012. 6 dez. 2011.

Spring Social Showcase. Github website. Disponível em: https://github.com/SpringSource/spring-social-samples/tree/master/spring-social-showcase . Último acesso em 23 jun. 2012

Task Execution and Scheduling. Disponível em http://static.springsource.org/spring/docs/3.0.x/reference/scheduling.html. Último acesso em 27 out 2012.

Use Case. Disponível em http://en.wikipedia.org/wiki/Use_case#Use_case_structure. Último acesso em 27 out 2012.

WADL – Web Application Description Language. Disponível em http://en.wikipedia.org/wiki/Web_Application_Description_Language. Último acesso em 27 out 2012.

WIKIPEDIA. Service Oriented Architecture. Disponível em: <http://en.wikipedia.org/wiki/Service_Oriented_Architecture> Acesso em: 19 jun. 2012