DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Dyfocus: Desenvolvimento do *Back-End* de um Aplicativo *Mobile* para *Smartphone*

Relatório submetido à Universidade Federal de Santa Catarina

como requisito para a aprovação na disciplina

DAS 5511: Projeto de Fim de Curso

Alexandre Costa Cordeiro

Florianópolis, fevereiro de 2014

Dyfocus: Desenvolvimento do *Back-End* de um Aplicativo *Mobile* para *Smartphone*

Alexandre Costa Cordeiro

Esta monografia foi julgada no contexto da disciplina
DAS5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação

Prof. *Rômulo Silva de Oliveira*

_____
Assinatura do Orientador

Banca Examinadora:

João Fernando Gomes de Oliveira
Orientador na Empresa

Prof. Rômulo Silva de Oliveira
Orientador no Curso

Prof. Ricardo José Rabelo
Responsável pela disciplina

Agradecimentos

Dedico este trabalho à minha família, que sempre se faz presente, mesmo na distância, em cada momento importante da minha vida; à minha namorada, companheira e amiga Camilla, que me dá apoio incondicional; aos meus sócios e co-fundadores Cássio, Marcelo e Victor, que há tanto tempo estão juntos comigo na busca da realização de um sonho; à toda família Cheesecake Labs, que é, de fato, a concretização desse sonho.

Aos Professores Rômulo e João Fernando, que fizeram de tudo para ajudar este trabalho a se concretizar.

Agradeço, de maneira especial e solene, ao meu amigo e sócio Cássio, pelas imensuráveis contribuições no desenvolvimento desta monografia.

Finalmente, agradeço a todos os meus demais amigos, sem os quais eu não seria capaz de realizar nada.

Resumo

Em um período no qual as aplicações para smartphones apresentam um crescimento considerável no mercado, surge a necessidade da criação de soluções que dêem suporte ao armazenamento de dados, troca de informações e forneçam funcionalidades extras que permitam ao usuário uma experiência agradável, rápida e livre de falhas. Neste contexto, servidores web, que antes se limitavam a fornecer páginas de hipertexto que permitissem a visualização de dados (estáticos ou dinamicamente carregados e renderizados), começam a se especializar na estruturação de dados e processamento dos mesmos, deixando que os *apps* se preocupem com a parte estética. Surgem, então, ferramentas como JSON (objetos javascript), cuja finalidade é apresentar estruturas complexas de dados de uma maneira compreensível, tanto para máquinas quanto para seres humanos. Paralelamente, surgem frameworks *web* e *APIs web*, que acessam banco de dados, estruturam as informações da maneira requisitada pelo *app* em objetos JSON e as transfere por protocolo HTTP. Este conceito, juntamente com a difusão da Internet, traz à tona definições como *Cloud Storage* e *Cloud Computing*, que consistem no armazenamento e processamento de grande parte dos dados que os usuários necessitam de maneira centralizada e remota, de modo que eles sejam disponibilizados em diversos dispositivos simultaneamente. Este trabalho trata da estruturação, projeto e implementação de um servidor *web* que dê esse tipo de suporte para um *app* desenvolvido para iPhone cuja funcionalidade principal é tirar fotos de uma mesma cena e diferentes pontos focais, bem como compartilhá-las no Facebook, criar comentários, curtidas e todo o aparato padrão de redes sociais.

Palavras-chave: smartphone, aplicações móveis, back-end, fotografia, pontos focais, rede social.

Abstract

The growth of smartphone applications' market share is nowadays undeniable. This brings along the need for establishing solutions which support, at the same time, reliable data storage and information exchange, as well as provide functionalities that allow users to have a pleasant, fast and ideally flawless user experience. In this context, web servers, which before were limited to offering hypertext pages that showed the (static or dinamically renderized) data, start to specialize in structuring information and processing it, leaving the aesthetic part to the apps themselves. To accomplish these new tasks, tools like JSON (Javascript objects) come up, allowing complex data structures to be presented in a both human- and machine-friendly way. At the same time, *web frameworks* and *web APIs* which access the database, organize the requested data in JSON objects in an almost seamless way become fundamental. This concept, along with the Internet popularization, brings up definitions such as Cloud Storage and Cloud Computing, which consists on storing and processing considerable part of the data needed by the users in a centralized and remote manner, so it can be available to different devices simultaneously. This work discusses the structuring, design and implementation of a web server which provides this cloud tools to an iPhone app whose core functionalities are to be able to take multiple pictures of the scene but with different focal points, as well as share them in Facebook, make comments and likes, along with all the default social network features.

Keywords: smartphone, mobile applications, back-end, photography, multiple focal points, social network.

Sumário

Índice de Anexos

Figuras

Tabelas

Capítulo 1: Introdução

1.1: Considerações Preliminares

Existente desde julho de 2012, a Cheesecake Labs é uma empresa de software fundada por quatro estudantes (três da graduação e um do mestrado) de Engenharia de Controle e Automação da Universidade Federal de Santa Catarina.

Criada com o preceito de desenvolver softwares inovadores, a Cheesecake tem seu foco principal em aplicações móveis, principalmente para as plataformas iOS (Apple) e Android (Google). A startup tem em seu portfólio uma série de aplicativos já desenvolvidos, sendoa maior parte deles em parceria com empresas do vale do silício, na Califórnia.

A decisão de tomar um rumo relacionado à Tecnologia de Informação e Comunicação (TIC) veio do anseio de utilizar todos os conhecimentos adquiridos no curso de Engenharia de Controle e Automação relacionados a metodologias de desenvolvimento de software, integração de sistemas de informação, sistemas distribuídos, fundamentos de bancos de dados, redes, inteligência artificial - tudo isso unido a uma vontade de empreender, inovar e imergir no mercado de aplicações mobile, que é extremamente novo e apresenta inúmeros cases de sucesso recentes (Instagram, Cinemagram, Vine, SoundCloud, Lumia, FocusTwist, Lytro, Ocarina, Photosynth e uma série de outras). É válido, ainda, destacar que os apps que até então escolhemos desenvolver tangem aspectos artísticos (imagem e música), tendo como modelos os casos supracitados.

Dentre os mais recentes desenvolvidos pela nossa empresa, destacam-se o defocus (já disponível na App Store, aplicativo que permite criar um formato de imagem que contenha diferentes distâncias ao plano focal, e que será abordado neste Projeto de Fim de Curso), bem como o TotalFocus (aplicativo capaz de tirar fotos inteiramente focadas) e o PitchShiftApp, aplicativo capaz de adicionar terças, quintas e tríades maiores musicais (técnica conhecida como Pitch Shifting) em qualquer gravação realizada pelo iPhone.

Os softwares são desenvolvidos em uma metodologia ágil, inspirados e orientados ao modelo MVC, tanto no front-end quanto no back-end, e o regime de trabalho da empresa é de total imersão: os maiores acionistas moram, atualmente, juntos e se dedicam 100% do seu tempo hábil na melhoria ou criação de novos apps e, mesmo, no aprendizado de novas tecnologias que possam agregar algo aos produtos finais.

Como a empresa é uma startup, a orientação do desenvolvimento foi realizada pelo professor Rômulo Silva de Oliveira, também orientador na Universidade Federal de Santa Catarina.

1.2: Organização desta monografia

Este trabalho foi organizado em seis capítulos principais. Primeiramente, esclarecem-se os objetivos do desenvolvimento desta monografia; em seguida, definem-se as ferramentas que serão utilizadas para sua realização; então, desenvolve-se o projeto de *back-end* do aplicativo, desde a instanciação do servidor até a estruturação de seu banco de dados; logo após, parte-se para a apresentação do desenvolvimento (implementação) *de facto*, com exemplos de código e imagens ilustrativas. Finalmente, mostram-se os resultados obtidos com o trabalho e as conclusões às quais o autor desta monografia chegou.

Capítulo 2: Objetivos

Desenvolvimento de um aplicativo para iPhone, integrado a rede sociais (Facebook), no estilo *Instagram* [36], que seja capaz de tirar fotos com múltiplos pontos focais e filtros. A interface deve permitir que o usuário escolha quais esses pontos serão e, no formato final, seja apresentado um conjunto de imagens no estilo "GIF" que alterne as fotos, dando a impressão de ser apenas uma imagem dinâmica – daí a origem do nome, *Dyfocus*.

Dentro deste projeto, coube ao autor desta monografia a parte de *back-end development*, i.e., criação, execução e manutenção de um servidor com arquitetura MVC [37], contendo banco de dados de usuários, armazenamento de imagens, website desenvolvido em linguagem de alto nível e toda a interface de comunicação com o aplicativo, de modo a fornecer todo o aparato para integração com redes sociais, oferecimento de recursos *cloud computing* [38], bem como a possibilidade de suportar todas as *features in-app* necessárias, como adicionar comentários, "curtir" fotos, etc.

Capítulo 3: Ferramentas e Conceitos Teóricos

Antes de apresentar o projeto e sua implementação, é importante fazer menção e listar as linguagens de programação, ferramentas e conceitos teóricos utilizados que guiaram e deram suporte no desenvolvimento do projeto (e produto) assim como foram fundamentais para a integração entre os diferentes módulos do aplicativo (*back-end* e *front-end*).

Nas próximas seções, as linguagens (Objective-c e Python), ferramentas (Xcode [17], TextMate [21] e JSON [24]), infra-estruturas (MySQL [29] e Amazon [34]), assim como as teorias abordadas (Orientação a objetos [10] e MVC [37]) serão apresentadas mais a fundo.

3.1: Linguagens

A escolha das linguagens de programação foi feita sem grandes dificuldade. Como o aplicativo foi desenvolvido para iPhone, a única linguagem de programação existente para se desenvolver no sistema operacional de *mobiles* da Apple (iOS [03]) é a linguagem denominada Objective-C, logo, o *front-end* foi todo desenvolvido em cima dessa linguagem. No servidor (*back-end)* optamos por uma das linguagens de programação mais atuais e de ponta, que oferece inúmeras bibliotecas e frameworks para os mais variados tipos de serviços e integração com outras tecnologias: Python [11].

Nos capítulos seguintes será dada uma introdução a essas linguagens.

3.1.1: Objective-C

A linguagem Objective-C é uma linguagem de programação reflexiva, orientada a objeto, que adiciona transmissão de mensagens no estilo Smalltalk [01] para o C [05]. É utilizada hoje, principalmente, no Mac OS X e GNUstep – dois ambientes baseados no padrão OpenStep [39] – e é a principal linguagem utilizada nos aplicativos estruturais NeXTSTEP [39], OPENSTEP [39] e Cocoa. Programas

genéricos em Objective-C que não façam uso destas bibliotecas também podem ser compilados por qualquer sistema suportado pelo gcc, que inclui um compilador Objective-C [06].

O Objective-C é uma camada muito fina construída sobre a linguagem C. Em outras palavras, é possível compilar qualquer programa C com um compilador Objective-C. A maior parte de sua sintaxe (incluindo pré-processamento, expressões, declaração e chamadas de funções) foi herdada da linguagem C, enquanto a sintaxe para os aspectos orientados a objetos foi criada para habilitar passagem de mensagens no estilo Smalltalk [01].

O modelo de orientação a objeto do Objective-C é baseado na passagem de mensagens para instâncias de objeto, em contraposição ao o modelo de programação estilo Simula, utilizado pelo C++ [40].

3.1.2: Python

Lançada pelo holandês Guido van Rossum em 1991, Python é uma linguagem de programação de tipagem dinâmica e forte. Definida como de alto nível, interpretada, imperativa e orientada a objetos, ela atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. O padrão de fato é a implementação CPython [12].

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus [11], embora muitas pessoas façam associação com o réptil do mesmo nome (pitão, em português).

A Python suporta a maioria das técnicas da programação orientada a objeto. Qualquer objeto pode ser usado para qualquer tipo, e o código funcionará enquanto haja métodos e atributos adequados. O conceito de objeto na linguagem é bastante abrangente: classes, funções, números e módulos são todos considerados objetos.

Também há suporte para metaclasses, polimorfismo, e herança (inclusive herança múltipla). Há um suporte limitado para variáveis privadas.

Python possui uma grande biblioteca padrão, geralmente citada como um dos maiores triunfos da linguagem, fornecendo ferramentas para diversas tarefas. Recentemente, a biblioteca Boost do C++ incluiu uma biblioteca para permitir a interoperabilidade entre as duas linguagens. Por conta da grande variedade de ferramentas fornecida pela biblioteca padrão, combinada com a habilidade de usar linguagens de nível mais baixo como C e C++, Python pode ser poderosa para conectar componentes diversos de software.

A biblioteca padrão conta com facilidades para escrever aplicações para a Internet, contando com diversos formatos e protocolos como MIME e HTTP. Também há módulos para criar interfaces gráficas, conectar em bancos de dados relacionais e manipular expressões regulares.

3.1.3: JSON

JavaScript Object Notation, popularmente conhecido como JSON [24], é um formato leve para intercâmbio de dados computacionais. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. JSON é um subconjunto da notação de objeto de JavaScript, mas seu uso não requer JavaScript exclusivamente.

O formato JSON foi originalmente criado por Douglas Crockford e é descrito no RFC 4627. O *media-type* oficial do JSON é “application/json” e a extensão é “.json” [24].

A simplicidade de JSON tem resultado em seu uso difundido, especialmente como uma alternativa para XML em AJAX. Uma das vantagens reivindicadas de JSON sobre XML como um formato para intercâmbio de dados neste contexto, é o fato de ser muito mais fácil escrever um analisador JSON. Em JavaScript mesmo, JSON pode ser analisado trivialmente usando a função *eval()*. Isto foi importante para a aceitação de JSON dentro da comunidade AJAX devido a presença deste recurso de JavaScript em todos os navegadores web atuais [26].

Na prática, os argumentos a respeito da facilidade de desenvolvimento e desempenho do analisador são raramente relevados devido aos interesses de segurança no uso de *eval()* e a crescente integração de processamento XML nos

navegadores web modernos. Por esta razão JSON é tipicamente usado em ambientes onde o tamanho do fluxo de dados entre o cliente e o servidor é de suma importância (daí seu uso por Google, Yahoo, etc., os quais servem milhões de usuários), onde a fonte dos dados pode ser explicitamente confiável, e onde a perda dos recursos de processamento XSLT no lado cliente para manipulação de dados ou geração da interface, não é uma consideração [25].

Enquanto JSON é frequentemente posicionado "em confronto" com XML, não é incomum ver tanto JSON como XML sendo usados na mesma aplicação. Por exemplo, uma aplicação no lado cliente a qual integra dados do Google Maps com dados atmosféricos através de SOAP, requer suporte para ambos formatos de dados [25].

Existe um crescente suporte para JSON através do uso de pequenos pacotes de terceiros. A lista de linguagens suportadas inclui: ActionScript, C/C++, C#, ColdFusion, Java, JavaScript, OCaml, Perl, PHP, ASP 3.0, Python, Rebol, Ruby, Lua, Progress [24].

```
{ "Aluno" : [
        { "nome": "João", "notas": [ 8, 9, 7 ]  },
        { "nome": "Maria", "notas": [ 8, 10, 7 ] },
        { "nome": "Pedro", "notas": [ 10, 10, 9 ] }
    ]
}
```

Figura 1 – Exemplo de um Objeto Json [24]

Pode-se observar pela figura 1 a simplicidade da sintaxe adotada em JSON, facilitando a troca de dados entre linguagens de diferentes tecnologias.

3.2: Frameworks

3.2.1: Django

Django é uma framework de aplicação web escrita em Python que segue o padrão de arquitetura MVC [37] (descrito na sequência). Ele é mantido pela Django Software Foundation [43], uma organização independente sem fins lucrativos.

O objetivo fundamental do Django é facilitar a criação de websites complexos fortemente relacionados com bases de dados. Ele enfatiza reusabilidade e conectividade de componentes, desenvolvimento rápido e o princípio DRY (*don't repeat yourself*) [43].

A linguagem Python é usada por toda a framework, mesmo para os arquivos de configuração e para a descrição dos modelos. A Django também provê uma interface administrativa de criação, leitura, atualização e remoção de dados, que é gerada dinamicamente através de *introspecção* e configurada através dos modelos *admin* (arquivo *admin.py*) [43].

Alguns websites famosos que utilizam tal *framework* são: Pinterest, Instagram, Mozilla, The Washington Times e o PBS – Public Broadcasting Service [43].

3.3: Ferramentas de Desenvolvimento

Nesse projeto, foram utilizados principalmente dois ambientes de desenvolvimento.

A IDE denominada Xcode [17], criada e mantida pela Apple, Inc. especialmente para os seus desenvolvedores, foi utilizada para o desenvolvimento do *front-end* do aplicativo, ou seja, toda a parte gráfica visualizada pelo smartphone e também os algoritmos internos processados pelo hardware do celular.

O lado do servidor (*back-end*) foi desenvolvido no editor de texto TextMate [22], também da Apple, propiciando um ambiente de desenvolvimento super leve e com uma considerável praticidade na manipulação de códigos e pastas do ambiente de trabalho [21].

Nos capítulos seguintes será feita uma breve apresentação dessas importantes ferramentas de desenvolvimento.

3.3.1: Xcode

O Xcode é um ambiente de desenvolvimento integrado da Apple, Inc. para gerenciamento de projetos relacionados com o sistema operacional de Mac OS X.

Ele possui ferramentas para o usuário criar e melhorar seus aplicativos. É um software poderoso e mais simples de utilizar para o desenvolvimento de aplicativos grandes [19].

A IDE já vem com as ferramentas necessárias para desenvolver aplicações para o Mac OS X, e suporta, por padrão, Objective-C e Apple-Script, que são linguagens de programação [19].

O Xcode tem um conjunto de "extras" para desenvolvimento, esses "extras" são chamados de SDK, e são fornecidos pela Apple Inc. no website de desenvolvimento de iOS deles. Empresas e desenvolvedores independentes devidamente cadastrados no programa para desenvolvedor iOS da Apple, chamado iOS Developer Program, podem distribuir os aplicativos na App Store [18].

3.3.2: TextMate

O TextMate é uma ferramenta de edição de texto puro para Mac OS com *highlight* de sintaxe, muito similar ao Notepad++ do Windows [21].

Com suporte para diversas linguagens, o TextMate permite ao programador encontrar e substituir palavras nos arquivos, recuo automático para ações comuns como colar texto e auto-emparelhamento dos suportes. Em sua janela principal, ele exibe linhas e colunas do texto ou programa, que por sua vez são alteradas por meio da coluna ao lado esquerdo [21].

Se necessário, o programa oferece suporte para que o usuário trabalhe como editor externo de arquivos remotos através de conexão FTP. Com suporte para mais de 50 idiomas, ele conta com mais de 20 gramáticas de programação, entre elas: ActionScript, C, C++, HTML, Java, JavaScript, LaTeX, Objective-C, OpenGL, Perl, PHP, Texto Plano, Python, Ruby, Ruby on Rails, Shell Script, SQL, XML, YAML e outras [22].

Além disso, o programa é relativamente leve e pode ser aberto em qualquer computador [22].

3.4: Database

Devido ao fato de o aplicativo estar imerso em uma estrutura de rede social, é preciso ter um complexo banco de dados para salvar as informações persistentes

relacionadas com cada usuário, por exemplo nome, email, senha, curtidas, comentários, imagens, dentre outras.

Para os dados em formato de texto, foi decido utilizar o sistema de gerenciamento de banco de dados mais difundido na internet, o MySQL. Porém, ainda precisaríamos de um banco de dados para o armazenamento de imagens, para esse propósito foi instanciado um servidor na Amazon Web Services (AWS S3) [34].

3.4.1: MySQL

O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês Structured Query Language) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo [30].

Entre os usuários do banco de dados MySQL estão: NASA, Friendster, Banco Bradesco, Dataprev, HP, Nokia, Sony, Lufthansa, U.S. Army, U.S. Federal Reserve Bank, Associated Press, Alcatel, Slashdot, Cisco Systems, Google e outros [30].

O MySQL foi criado na Suécia por suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius, que têm trabalhado juntos desde a década de 1980. Hoje seu desenvolvimento e manutenção empregam aproximadamente 400 profissionais no mundo inteiro, e mais de mil contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito dele [28].

No dia 16 de Janeiro de 2008, a MySQL AB, desenvolvedora do MySQL foi adquirida pela Sun Microsystems, por US$ 1 bilhão, um preço jamais visto no setor de licenças livres. No dia 20 de Abril de 2009, foi anunciado que a Oracle compraria a Sun Microsystems e todos o seus produtos, incluindo o MySQL. Após investigações da Comissão Europeia sobre a aquisição para evitar formação de monopólios no mercado a compra foi autorizada e hoje a Sun faz parte da Oracle [30].

O sucesso do MySQL deve-se em grande medida à fácil integração com o PHP incluído, quase que obrigatoriamente, nos pacotes de hospedagem de sites da Internet oferecidos atualmente. Empresas como Yahoo! Finance, MP3.com, Motorola, NASA, Silicon Graphics e Texas Instruments usam o MySQL em

aplicações de missão crítica. A Wikipédia é um exemplo de utilização do MySQL em sites de grande audiência [30].

O MySQL hoje suporta Unicode, Full Text Indexes, replicação, Hot Backup, GIS, OLAP e muitos outros recursos de banco de dados [29].

Algumas características notáveis que nos levaram a escolher o MySQL como banco de dados:

- Portabilidade (suporta praticamente qualquer plataforma atual);
- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, C#, Visual Basic, Python, Perl, PHP, ASP e Ruby);
- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de novos hardwares;
- Facilidade no manuseio;
- É um Software Livre com base na GPL;
- Contempla a utilização de vários Storage Engines como MyISAM, InnoDB, Falcon, BDB, Archive, Federated, CSV, Solid, etc.;
- Suporta controle transacional;
- Suporta Triggers;
- Suporta Cursors (Non-Scrollable e Non-Updatable);
- Suporta Stored Procedures e Functions;
- Replicação facilmente configurável;
- Interfaces gráficas (MySQL Toolkit) de fácil utilização cedidos pela MySQL Inc.

3.5: Web Server, Interfaces e Servidores

3.5.1: Apache HTTP Server

O servidor Apache (ou Servidor HTTP Apache, em inglês: Apache HTTP Server, ou simplesmente: Apache) é o mais bem sucedido servidor web livre [33]. Foi criado em 1995 por Rob McCool, então funcionário do NCSA (*National Center for Supercomputing Applications*). Numa pesquisa realizada em dezembro de 20071 , foi constatado que a utilização do Apache representa cerca de 47.20% dos servidores ativos no mundo. Em maio de 2010 2 , o Apache serviu aproximadamente

54,68% de todos os sites e mais de 66% dos milhões de sites mais movimentados. É a principal tecnologia da Apache Software Foundation, responsável por mais de uma dezena de projetos envolvendo tecnologias de transmissão via web, processamento de dados e execução de aplicativos distribuídos [31].

O servidor é compatível com o protocolo HTTP versão 1.13. Suas funcionalidades são mantidas através de uma estrutura de módulos, permitindo inclusive que o usuário escreva seus próprios módulos — utilizando a API do software [32].

É disponibilizado em versões para os sistemas Windows, Novell Netware, OS/2 e diversos outros do padrão POSIX (Unix, Linux, FreeBSD, dentre outros) [32].

3.5.2: WSGI

O Web Server Gateway Interface (WSGI) define uma interface simples e universal entre servidores web e aplicações web ou frameworks para a linguagem de programação Python [35].

O WSGI possui dois lados: o lado "servidor" ou "gateway" e o lado "aplicação" ou "framework". Para processar uma requisição WSGI, o lado servidor fornece informações de ambiente e uma função de *callback* para o lado aplicação. A aplicação processa a requisição e retorna a resposta para o lado servidor usando a função de *callback* que o lado servidor forneceu [35].

O chamado *middleware* WSGI implementa os dois lados da API. Desta forma, ele pode intermediar a comunicação entre um servidor WSGI e uma aplicação WSGI: o middleware age como uma aplicação de algum ponto de vista do servidor WSGI e como um servidor de algum ponto de vista da aplicação WSGI [35].

3.5.3: Amazon Web Services

Amazon Web Services (AWS) é um conjunto de serviços de computação remota (também chamados web services) que juntos, constituem uma plataforma de computação em nuvem, proporcionada através da Internet pela Amazon.com. Os serviços mais populares são o Amazon EC2 e o Amazon S3 [34].

Desde o início de 2006, a Amazon Web Services (AWS) tem fornecido às empresas de todos os portes uma infraestrutura de serviços web com plataforma em nuvem. Com a AWS você pode calcular o poder de computação, armazenamento e outros serviços relacionados para acessar o pacote de serviços de infraestrutura de TI elástica como exigido por seus negócios. Com a AWS, é possível ter a flexibilidade de escolher qualquer plataforma de desenvolvimento ou modelo de programação que se adapte aos problemas que o usuário está tentando resolver. [34].

Usando a Amazon Web Services, um website de comércio eletrônico pode resistir facilmente a uma demanda imprevista; uma empresa farmacêutica pode "alugar" poder de computação para executar simulações em grande escala; uma empresa de mídia pode disponibilizar um numero ilimitado de vídeos, música e muito mais; e uma empresa pode implantar serviços e treinamento utilizando largura de banda para sua força de trabalho móvel [34].

3.6: Metodologia de Desenvolvimento

O desenvolvimento do aplicativo, tanto do *front-end (iPhone)* quanto do *back-end (servidor)*, foi feito em cima de dois grandes pilares da teoria de programação de software:

- Programação Orientada a Objetos, e
- Metodologia MVC

Nos capítulos subsequentes abordaremos ambas as teorias empregadas.

3.6.1: Orientação a Objetos

A orientação a objetos é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos [09].

Em alguns contextos, prefere-se usar modelagem orientada ao objeto, em vez de programação. De fato, o paradigma "orientação a objeto", tem bases conceituais e origem no campo de estudo da cognição, que influenciou a área de inteligência artificial e da linguística, no campo da abstração de conceitos do mundo real. Na

qualidade de método de modelagem, é tida como a melhor estratégia para se eliminar o "gap semântico", dificuldade recorrente no processo de modelar o mundo real do domínio do problema em um conjunto de componentes de software que seja o mais fiel na sua representação deste domínio. Facilitaria a comunicação do profissional modelador e do usuário da área alvo, na medida em que a correlação da simbologia e conceitos abstratos do mundo real e da ferramenta de modelagem (conceitos, terminologia, símbolos, grafismo e estratégias) fosse a mais óbvia, natural e exata possível [10].

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. C++, C♯, VB.NET, Java, Object Pascal, Objective-C, Python, SuperCollider, Ruby e Smalltalk são exemplos de linguagens de programação orientadas a objetos. ActionScript, ColdFusion, Javascript, PHP (a partir da versão 4.0), Perl (a partir da versão 5) e Visual Basic (a partir da versão 4) são exemplos de linguagens de programação com suporte a orientação a objetos [41].

A orientação a objetos possui vários conceitos essenciais, listaremos aqui os principais:

- **Classe:** representa um conjunto de objetos com características afins. Uma classe define o comportamento dos objetos através de seus métodos, e quais estados ele é capaz de manter através de seus atributos;
    - Subclasse é uma nova classe que herda características de sua(s) classe(s) ancestral(is);
- **Objeto / instância** de uma classe. Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos;
- **Atributo** são características de um objeto. Basicamente a estrutura de dados que vai representar a classe. Por sua vez, os atributos possuem valores. Por exemplo, o atributo cor pode conter o valor azul. O conjunto de valores dos atributos de um determinado objeto é chamado de estado;

- **Método** definem as habilidades dos objetos. Um método em uma classe é apenas uma definição. A ação só ocorre quando o método é invocado através do objeto. Dentro do programa, a utilização de um método deve afetar apenas um objeto em particular. Normalmente, uma classe possui diversos métodos;
- **Mensagem** é uma chamada a um objeto para invocar um de seus métodos, ativando um comportamento descrito por sua classe. Também pode ser direcionada diretamente a uma classe (através de uma invocação a um método estático);
- **Herança** (ou generalização) é o mecanismo pelo qual uma classe (sub-classe) pode estender outra classe (super-classe), aproveitando seus comportamentos (métodos) e variáveis possíveis (atributos). Há herança múltipla quando uma sub-classe possui mais de uma super-classe;
- **Associação** é o mecanismo pelo qual um objeto utiliza os recursos de outro. Pode tratar-se de uma associação simples "usa um" ou de um acoplamento "parte de". Por exemplo: Um humano usa um telefone. A tecla "1" é parte de um telefone;
- **Encapsulamento** consiste na separação de aspectos internos e externos de um objeto. Este mecanismo é utilizado amplamente para impedir o acesso direto ao estado de um objeto (seus atributos), disponibilizando externamente apenas os métodos que alteram estes estados;
- **Abstração** é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software;
- **Polimorfismo** consiste em quatro propriedades que a linguagem pode ter (atente para o fato de que nem toda linguagem orientada a objeto tem implementado todos os tipos de polimorfismo):
    - Universal:
        - Inclusão: um ponteiro para classe mãe pode apontar para uma instância de uma classe filha;

        - Paramétrico: se restringe ao uso de templates (C++, por exemplo) e generics (Java/C♯);
    - Ad-Hoc:
        - Sobrecarga: duas funções/métodos com o mesmo nome mas assinaturas diferentes;
        - Coerção: a linguagem que faz as conversões implicitamente (como por exemplo, atribuir um int a um float em C++, isto é aceito mesmo sendo tipos diferentes pois a conversão é feita implicitamente);
- **Interface** (ciência da computação) é um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface;
- **Pacotes** (ou Namespaces) são referências para organização lógica de classes e interfaces [10];

3.6.2: Metodologia MVC

Model-view-controller (MVC) é um modelo de arquitetura de software que separa a representação da informação da interação do usuário com ele. O modelo (*model*) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (*view*) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. O controlador (*controller*) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos [37].

Apesar de desenvolvida originalmente para computação pessoal, o MVC foi amplamente adaptado como uma arquitetura para as aplicações *World Wide Web* em todas as linguagens de programação maiores. Muitos *frameworks* de aplicação comerciais e não comerciais foram criados, que aplicam o modelo. Estes *frameworks* variam em suas interpretações, principalmente no modo que as responsabilidades MVC são divididas entre o cliente e servidor [37].

Os *frameworks web* MVC mais recentes levam uma abordagem de *thin client* que colocou quase o modelo, a visão e a lógica do controlador inteiros no servidor. Nesta abordagem, o cliente envia requisições de *hiperlink* ou entrada deformulário ao controlador e então recebe uma página web completa e atualizada (ou outro documento) da visão. O modelo existe inteiramente no servidor. Como as tecnologias de cliente amadureceram, frameworks como JavaScriptMVC e Backbone foram criados o que permite que os componentes MVC executem parcialmente no cliente (ver também AJAX [23]) [37].

Um caso prático é uma aplicação web em que a visão é um documento HTML (ou derivado) gerado pela aplicação. O controlador recebe uma entrada *GET* ou *POST* após um estímulo do utilizador e decide como processá-la, invocando objetos do domínio para tratar a lógica de negócio, e por fim invocando uma visão para apresentar a saída [37].

Com o aumento da complexidade das aplicações desenvolvidas, sempre visando à programação orientada a objeto, torna-se relevante a separação entre os dados e a apresentação das aplicações. Desta forma, alterações feitas no layout não afetam a manipulação de dados, e estes poderão ser reorganizados sem alterar o layout [37].

Esse padrão resolve este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois: o controlador [37].

Além de dividir a aplicação em três tipos de componentes, o desenho MVC define as interações entre eles. As próximas sub-seções expõe essas interações [37].

3.6.3: Model

Um modelo (*model*) notifica suas visões e controladores associados quando há uma mudança em seu estado. Esta notificação permite que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis. Uma implementação passiva do MVC monta estas notificações, devido à aplicação não necessitar delas ou a plataforma de software não suportá-las [37].

3.6.4: View

A visão (*view*) solicita do modelo a informação que ela necessita para gerar uma representação de saída [37].

3.6.5: Controller

Um controlador (*controller*) pode enviar comandos para sua visão associada para alterar a apresentação da visão do modelo (por exemplo, percorrendo um documento). Ele também pode enviar comandos para o modelo para atualizar o estado do modelo (por exemplo, editando um documento).

A figura abaixo exemplifica bem as interações entre os diferentes componentes da metodologia MVC [37]:

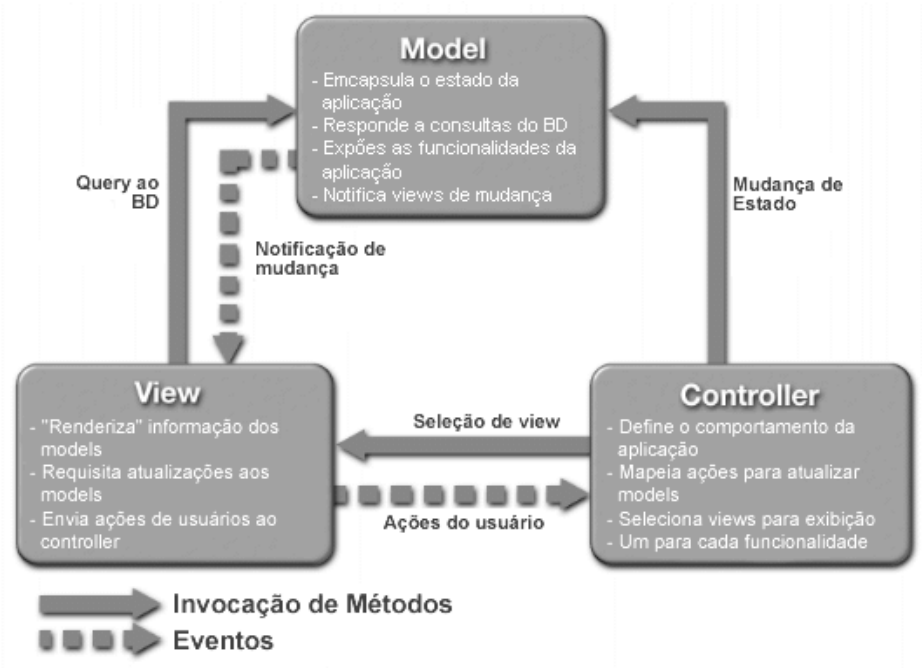

Figura 2 – Representação gráfica do modelo MVC [37]

Capítulo 4: Projeto do Backend do Dyfocus

Neste capítulo, serão contempladas as etapas de desenvolvimento do back-end do Dyfocus, desde a criação de sua estrutura até a implementação de algumas algumas de suas principais requisições.

O trabalho descrito ao decorrer deste capítulo foi completamente realizado pelo autor da monografia, desde a definição da infra-estrutura até a última iteração da esquematização do banco de dados relacional.

4.1: Instanciação do Servidor

Antes de desenvolver a estrutura do servidor, é necessário definir quais as configurações da máquina, de modo a ter um conhecimento das limitações de hardware/software.

Pesquisando os melhores e mais utilizados serviços web da atualidade, é notável que grande parte dos websites de grande porte (que não têm uma estrutura para manter seu próprio hardware, obviamente) utiliza dos serviços web providos pela Amazon – os Amazon Web Services, popularmente conhecidos como AWS.

Os AWS oferecem dois serviços que são interessantes para o projeto Dyfocus:

- **EC2 (Elastic Compute Cloud) –** Um serviço que permite um desenvolvimento escalável de aplicações, provendo um web service através do qual um usuário pode dar *boot* a uma *Amazon Machine Image* para criar uma máquina virtual, contendo qualquer software desejado. É possível escolher o sistema operacional, assim como as configurações de hardware;
- **S3 (Simple Storage Service) –** Como o próprio nome diz, é um serviço de armazenamento na nuvem, acessível através de interfaces de web services (REST, SOAP, etc). É ideal para usar como um *bucket* de armazenamento de fotos para o aplicativo Dyfocus.

Uma vez definidos tais serviços, é preciso definir quais as configurações para a instância de cada serviço.

4.1.1: EC2 (Elastic Compute Cloud)

4.1.1.1: Hardware

Para o serviço EC2, existem as seguintes configurações de hardware [44]:

Size	ECUs	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance
t1.micro	up to 2	1	0.613	EBS only	-	Very Low
m1.small	1	1	1.7	1 x 160	-	Low
m1.medium	2	1	3.7	1 x 410	-	Moderate
m1.large	4	2	7.5	2 x 420	Yes	Moderate
m1.xlarge	8	4	15	4 x 420	Yes	High
m3.medium	3	1	3.75	1 x 4 (SSD)	-	Moderate
m3.large	6.5	2	7.5	1 x 32 (SSD)	-	Moderate
m3.xlarge	13	4	15	2 x 40 (SSD)	Yes	Moderate
m3.2xlarge	26	8	30	2 x 80 (SSD)	Yes	High
m2.xlarge	6.5	2	17.1	1 x 420	-	Moderate
m2.2xlarge	13	4	34.2	1 x 850	Yes	Moderate
m2.4xlarge	26	8	68.4	2 x 840	Yes	High
hi1.4xlarge	35	16	60.5	2 x 1024 (SSD)	-	10 Gigabit

Size	ECUs	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance
hs1.8xlarge	35	16	117	24 x 2048	-	10 Gigabit
c1.medium	5	2	1.7	1 x 350	-	Moderate
c1.xlarge	20	8	7	4 x 420	Yes	High
c3.large	7	2	3.75	2 x 16 (SSD)	-	Moderate
c3.xlarge	14	4	7.5	2 x 40 (SSD)	Yes	Moderate
c3.2xlarge	28	8	15	2 x 80 (SSD)	Yes	High
c3.4xlarge	55	16	30	2 x 160 (SSD)	Yes	High
c3.8xlarge	108	32	60	2 x 320 (SSD)	-	10 Gigabit

Tabela 1 - Diversas configurações de Instâncias EC2 dos AWS

Para a escolha da configuração, levou-se em conta que o EC2 possui uma ferramenta de exportação de *imagem da instância (.ami)*, que permite a migração de hardware de maneira muito rápida. Sendo assim, não existe necessidade de escolher um servidor que suporte um número grande de conexões e tenha uma grande capacidade de processamento simultâneo, uma vez que ele servirá apenas para testes e para um número pequeno de usuários a curto prazo.

No entanto, é desejável que a memória RAM não seja mínima, uma vez que o consumo de memória do *Django*, que lida com cópias de tabelas do MySQL concorrentemente, é desconhecido, e ter uma queda no serviço por conta de falta de memória é indesejável.

Dessa maneira, escolhe-se instanciar uma máquina virtual do tipo *m1.medium*, que conta com apenas 2 unidades de processamento, 1 CPU virtual , mas 3.7 GiB de memória RAM.

4.1.1.2: Software

A essa altura, a única decisão a ser tomada em termos de software tem a ver com o sistema operacional desejado. Por questões de facilidade, deseja-se alguma distribuição de Linux que seja baseada em Debian, de modo que se tenha disponível a *aptitude*, ferramenta de front-end para o *Advanced Packaging Tool* (APT) [45]. Por questões de familiaridade, opta-se por usar a distribuição Ubuntu em sua versão LTS (*long-term support*), para a qual será dado suporte por tempo razoável até que o aplicativo já tenha uma base sólida.

4.1.1.3: Segurança de Acesso

Para acesso à instância, é preciso registrar a private key dos computadores que desejam ter acessos na lista de permissões do servidor, ou através da utilização de um arquivo *.pem*, gerado na instanciação. O acesso deve ser feito via SSH (*Secure Shell*).

De modo a permitir esse acesso, bem como permitir que o servidor possa oferecer serviços web e, finalmente, para permitir acesso fácil ao futuro banco de dados, foram desbloqueadas as seguintes portas para acesso externo:

- 22 (SSH);
- 80 (HTTP);
- 3306 (MySQL).

4.1.2: S3 (Simple Storage Service)

Para a instância S3, não existem opções de hardware. O serviço provê armazenamento ilimitado, com a simples restrição de que um arquivo único não pode ter mais de 5 TB.

4.2: Estrutura do Servidor

Uma vez definido o hardware e o acesso à máquina que armazenará o back-end do Dyfocus, propõe-se uma estrutura de servidor (vide Figura 1), juntamente com a explicação de cada um dos módulos referidos no diagrama.

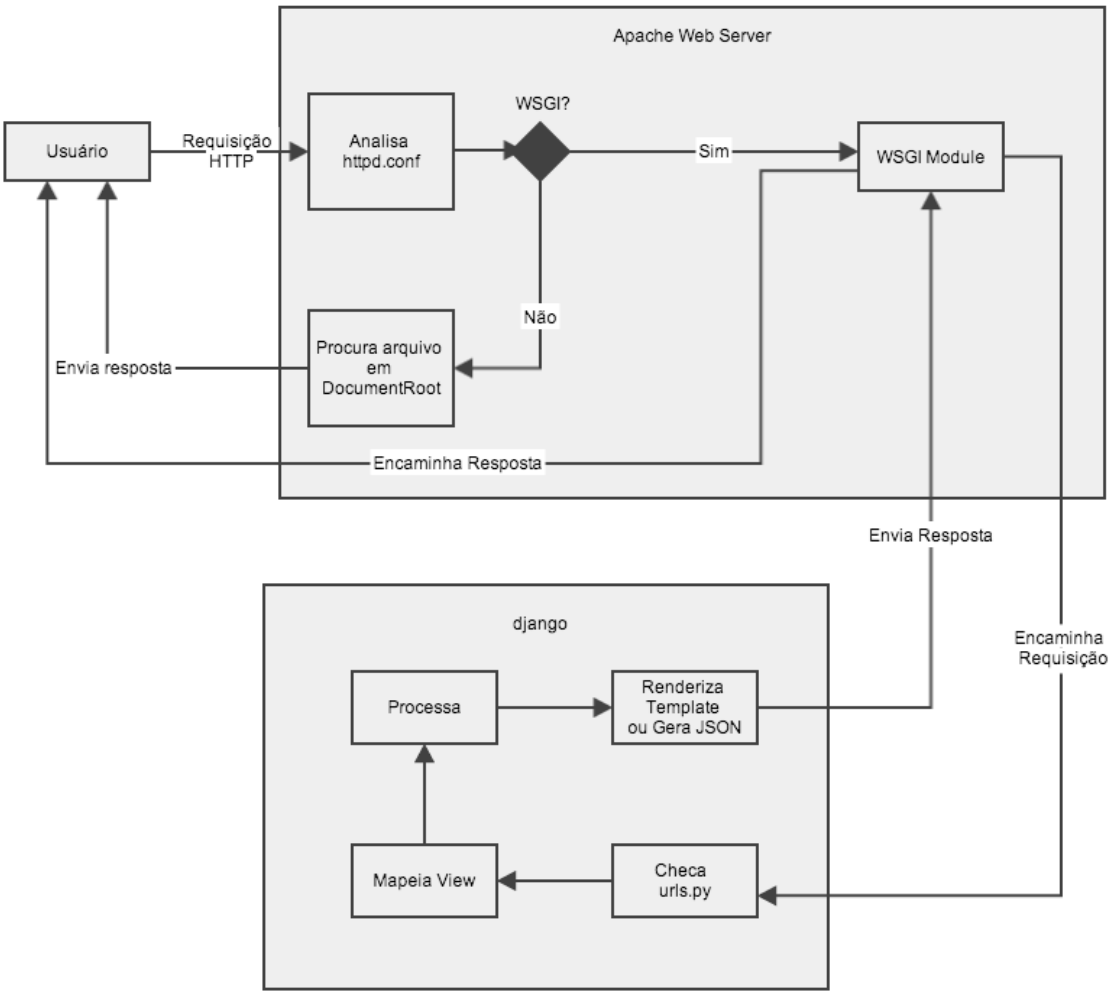

Figura 1 - Estrutura do servidor

4.2.1: Apache HTTP Server

O Apache é o aplicativo de web server mais utilizado e consagrado na Internet [46]. No contexto desse projeto, ele é responsável por lidar com todas as requisições enviadas através da porta 80.

Em um website que conta com uma estrutura convencional, i.e., com pouco processamento no back-end, o Apache é, também, destino final das requisições HTTP. Em outras palavras, é, também, o Apache que processa a requisição e serve os arquivos.

Em um contexto um pouco mais complexo, como é o caso da linguagem PHP, muito utilizada até hoje, o processador PHP é instalado como um módulo do

próprio Apache. As requisições chegam ao web server, ele as redireciona para seu módulo de PHP, que as processa e devolve como um conjunto de hipertextos estáticos.

Tal exemplo ilustra bem a definição de processamento *server-side*: um usuário pede ao servidor uma página (chamemos de *pagina.php*). O arquivo pagina.php é um híbrido de HTML e de códigos em PHP, que não interessam ao usuário – o mesmo só deseja o conteúdo de tal página após o processamento.

Sendo assim, ao receber a requisição pelo arquivo */pagina.php*, o usuário deseja obter o arquivo */pagina.html*, que nada mais é que o arquivo PHP com as tags PHP processadas e convertidas em tags estáticas HTML [47].

A Figura 2 ilustra tal situação:

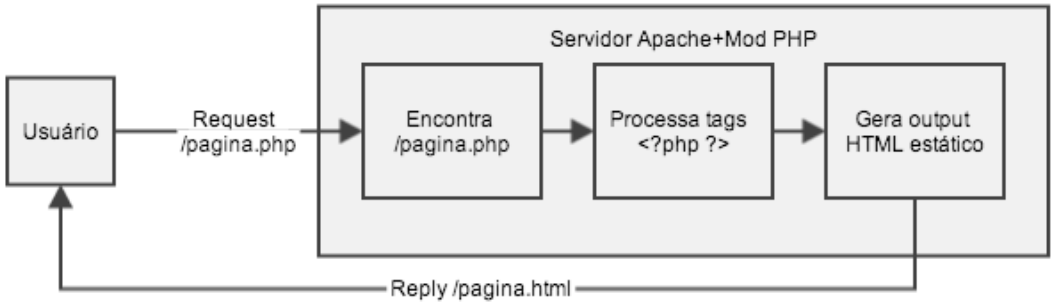

Figura 2 - Estrutura de um servidor Apache+PHP

Esse modelo de processamento *server-side* entra em contraposição com os do tipo *client-side*, como é o caso do Javascript [48].

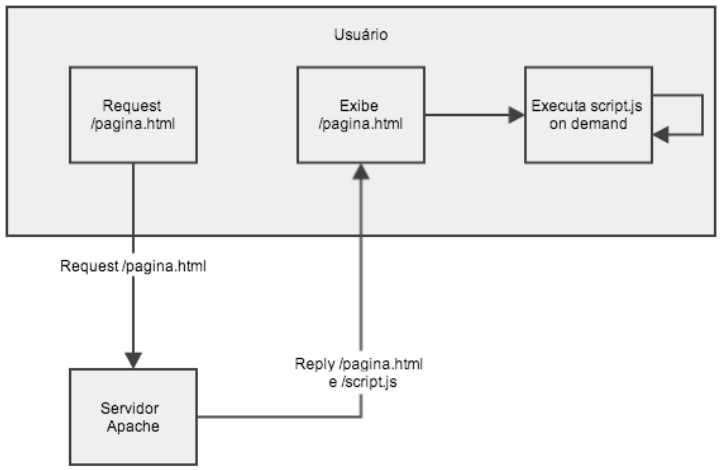

Figura 3 - Ilustração do Processamento Client-side

Como se pode notar pela Figura 3, os códigos Javascript são baixados juntamente com o arquivos HTML. O programa em Javascript é, então, interepretado em *runtime* e altera dinamicamente a página (DOM, *Document Object Model*) de acordo com sua implementação.

A estrutura back-end do Dyfocus, assim como no exemplo do PHP, também conta com processamento de arquivos de hipertexto *server-side*. Além de poder centralizar o acesso à base de dados e deixar que processamentos complexos sejam realizados em uma máquina conhecida, o processamento no servidor faz com que os usuários não tenham acesso à implementação dos programas do servidor, o que é consideravelmente interessante do ponto de vista de desenvolvimento de produto.

A grande diferença da relação Apache-PHP e Apache-django é que a última necessita de um elemento intermediário, uma vez que o django é um programa inteiro, e não um conjunto de fragmentos de código que podem ser executados independentemente. Dessa maneira, é preciso que haja um módulo/add-on do Apache capaz de compilar uma instância django (em Python), receber as requisições que chegam ao web server e redirecioná-las para ela [35].

Este módulo é denominado WSGI e será abordado no próximo tópico.

4.2.2: WSGI

Como explicado na introdução das ferramentas utilizadas, o WSGI (*Web-Server Gateway Interface)* é uma interface universal entre web-servers, web-applications ou frameworks para a linguagem Python. Ele foi criado com o intuito de permitir que frameworks web em Python como a *Django* (utilizada neste trabalho) sejam capazes de ser escaláveis e possam ser conectadas a web-servers mais consolidados e seguros, como o Apache.

Para integrar o WSGI ao Apache, foi preciso instalar o mod-wsgi (módulo wsgi) no servidor , bem como alterar as configurações do webserver no arquivo *httpd.conf*:

```
WSGIScriptAlias / /home/ubuntu/dyfocus/dyfocus/wsgi.py
WSGIPythonPath /home/ubuntu/dyfocus/

<Directory /home/ubuntu/dyfocus/dyfocus>
<Files wsgi.py>
Order deny,allow
Allow from all
</Files>
</Directory>
```

Bem como o arquivo wsgi.py, que utiliza o módulo *wsgi* do *django.core* para adaptar sua interface à WSGI.

```
import os

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "dyfocus.settings")

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

E, finalmente, nas configurações do django:

```
# Python dotted path to the WSGI application used by Django's runserv
er.
WSGI_APPLICATION = 'dyfocus.wsgi.application'
```

4.2.3: Django

O papel do Django no servidor é ser, de fato, o processador de back-end. A framework é organizada de acordo com a estrutura MVC (model-view-controller) [37]. No entanto, existem algumas diferenças na nomenclatura utilizada pelos seus desenvolvedores, que será clarificada (e adotada) de modo a não gerar ambiguidades ao longo do desenvolvimento deste projeto.

Segundo a interpretação dos desenvolvedores Django, “view” descreve os dados que são apresentados para o usuário. Ela não se limita, no entanto, a dizer *como* eles serão exibidos, mas *quais* dados serão exibidos. Dessa maneira, a maior parte do processamento programado usando Django se encontra nas denominadas *views*, em vez de em *controllers*.

Ainda segundo os desenvolvedores, isso é criado de modo a deixar clara a distinção entre conteúdo e apresentação do conteúdo – e isso não é claro no modelo MVC.

Sendo assim, o *controller* passa a ser a própria framework, que, internamente, lida com requisições, as processa e gera a devida resposta, na forma de um template. É dito que a Django trabalha com uma estrutura MTV: *model*, *template* e *view*:

- Camada *Model*: também denominada camada de abstração, é responsável por estruturar e manipular os dados da aplicação web. É esta camada a responsável pela integração/interface com o sistema de gerenciamento de banco de dados MySQL, de modo a armazenar os dados desejados e realizar *queries* para obtê-los de maneira estruturada;
- Camada *View*: o conceito de *views* encapsula a lógica responsável por processar a requisição de um usuário e retornar a resposta;
- Camada *Template*: provê uma sintaxe padrão de templates (muito utilizada por web designers) para renderizar a informação para que ela possa ser apresentada ao usuário.

De maneira breve, o Django funciona da seguinte maneira (para uma requisição já desenvolvida, testada e funcional):

1. Recebe uma requisição HTTP através de uma url, que ele comparará com todas as expressões regulares dadas no arquivo *urls.py*;
2. Encontra o método (view) correspondente na árvore de diretórios do Django (por *default*, no arquivo *views.py*);
3. Executa o método, o que geralmente consiste em:

- Fazer queries no banco de dados para encontrar as informações desejadas;
- Filtrar/modificar os resultados para que eles sejam exibidos da maneira desejada;
- Encontrar o template correspondente e renderizá-lo com esses dados;

4. Responder à requisição com um arquivo HTML (template renderizado) ou com um objeto JSON.

4.2.4: MySQL

O sistema de gerenciamento de banco de dados (SGBD) MySQL é adotado neste projeto por funcionar bem com o Django, além de ser (em julho de 2003) o segundo SGBD open-source mais utilizado no mundo, ficando atrás apenas do Sqllite, que é embutido em todos os dispositivos com sistema operacional Android ou iOS.

O uso do MySQL é bem sucedido em vários aplicativos Web com grande volume de dados, como Facebook [49] e Wikipedia [50].

4.3: Estrutura do Banco de Dados

4.3.1: Análise Preliminar

Antes de projetar a camada de persistência do aplicativo web, é preciso ter em mente quais tipos de dados serão armazenados na base da dados. É importante ressaltar que o Dyfocus visa a ter todo o seu armazenamento feito na nuvem, ou seja, nenhuma foto tirada será gravada automaticamente no aparelho celular do usuário.

Este conceito visa a possibilitar que o aplicativo se torne, em algum momento, uma rede social, com todo o aparato para que usuários possam visualizar fotos de seus amigos, curti-las, comentá-las, etc.

Tendo isso em mente, o back-end do aplicativo Dyfocus, em sua concepção mais primitiva, precisa ser capaz de:

- Reconhecer um usuário, i.e., perceber quem está acessando o banco de dados e associar isso a uma conta, de maneira transparente ou não;
- Armazenar fotos de um usuário (ou mesmo urls referente a elas), de modo a possibilitar que elas sejam reavidas quando o usuário voltar ao aplicativo;
- Criar coleções de fotos, ou seja, grupos de fotos que pertencem a uma mesma FOF (*focus-oriented frames*), que é um conjunto de fotos abordando o mesmo conteúdo, mas com pontos focais diferentes.

Esse grupo de funcionalidades primárias daria origem a, basicamente, três modelos (classes Python-django) na base de dados:

- Modelo **User**: reponsável pelo armazenamento de dados do usuário. Como atributos do usuário, em um primeiro momento, foram definidos como necessários o armazenamento do UDID (*Unique Device ID*, um identificador único que cada dispositivo da Apple tem) e da data de registro na base de dados.;
- Modelo **Frame**: é a menor divisão de uma FOF. Essencialmente, é uma entidade que armazena a URL de uma imagem, seu número de sequência, a posição de seus pontos focais e aponta para uma instância de FOF;
- Modelo **FOF**: é um conjunto de Frames, que armazena o seu autor, tem um tamanho (número de frames associadas) e uma data de publicação.

As tabelas inicias do MySQL ficam definidas da seguinte forma:

```
User (id[pk], device_id, pub_date);
```

```
Frame (id[pk], url, fof_id[fk], index, focal_point_x, focal_point_y);
```

```
FOF (id[pk], user_id[fk], size, pub_date);
```

E as suas relações:

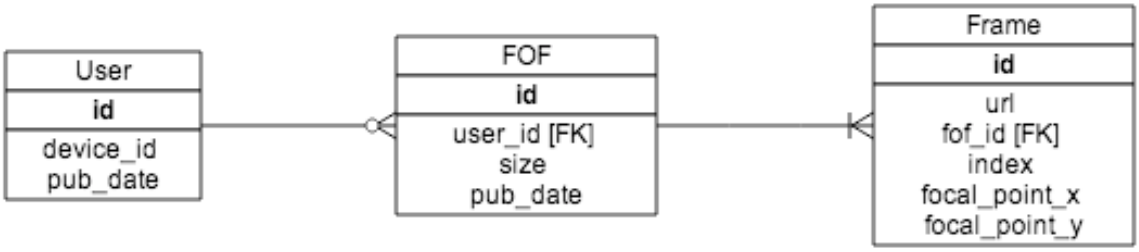

Figura 4 – Relação entre entidades do banco de dados

Traduzindo estas relações para a estrutura de códigos django (que criará as tabelas no MySQL automaticamente), temos:

```
import datetime
from django.utils import timezone
from django.db import models

class User(models.Model):
    device_id = models.CharField(max_length=200, null=True)
    pub_date = models.DateTimeField('date published')

    def __unicode__(self):
        return self.name


class FOF(models.Model):
    user = models.ForeignKey(User)
    size = models.IntegerField()
    pub_date = models.DateTimeField('date published')

    def __unicode__(self):
        return self.name


class Frame(models.Model):
    url = models.CharField(max_length=200)
    fof = models.ForeignKey(FOF)
    index = models.IntegerField()
    focal_point_x = models.IntegerField()
    focal_point_y = models.IntegerField()

    def __unicode__(self):
        return self.url

```

Com esta estrutura, já é possível construir as *views* que acessem as entidades armazenadas no banco de dados, bem como os templates. No entanto, após uma série de pequenos testes/implementações (contemplados no Capítulo 6), é possível concluir que esta estrutura possui falhas de escalabilidade, que nos leva à necessidade de uma melhor elaboração das entidades na base de dados.

4.3.2: Análise Avançada – Primeira Iteração

Após criar métodos básicos para criação de fotos (contemplados no Capítulo 6), concluiu-se que a estrutura do Banco de Dados proposta na seção anterior possui uma falha de escalabilidade intrínseca:

- Um usuário pode ter mais de um dispositivo (um iPhone e um iPod Touch, ou um iPhone e um iPad, por exemplo) e querer ter acesso a suas fotos em ambos os dispositivos.

- Um usuário pode migrar de um dispositivo para outro (adquirir um modelo mais novo de um iPhone, por exemplo).

Da maneira através da qual a base de dados foi estruturada, um usuário tem uma relação 1:1 (de um para um) com um dispositivo. O uso do app por outro dispositivo implica na criação de uma nova conta de usuário.

Para solucionar este problema, é necessário definir uma solução que torne a relação usuário-dispositivo do tipo ‘um para muitos’.

Com este fim, cria-se uma nova tabela relacional:

- Modelo **Device**: reponsável por descrever uma relação entre uma instância do modelo **User** com um UDID, de modo que cada usuário possa ser relacionado com vários UDIDs.

  A tabela apresenta, então, a seguinte estrutura:

  `Device` `(id[pk], user_id[fk], device_id);`

  As novas relações do banco de dados se tornam:

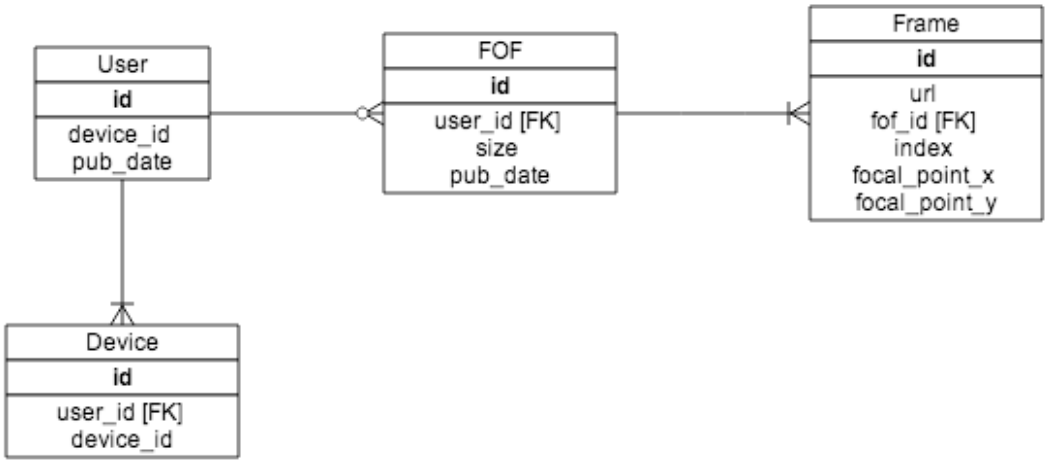

Figura 5 - Adição da entidade Device no banco de dados

Como é possível notar, o parâmetro *device_id* foi mantido no usuário, por questões de compatibilidade com as versões anteriores. Foi criado um código de mapeamento, de modo a migrar os dados de dispositivo já existentes na tabela de usuário para a tabela dedicada a essa relação usuário-dispositivo.

É adicionado ao arquivo *models.py* a interface para o modelo *Device*:

```
1. class Device(models.Model):
2.     user = models.ForeignKey(User)
3.     device_id = models.CharField(max_length=200)
```

Após a implementação dessa solução, grande parte dos problemas estava resolvida: um usuário era capaz de ter vários dispositivos. No entanto, ainda restava um outro problema de escalabilidade.

4.3.3: Análise Avançada – Segunda Iteração

Após resolver o problema de um usuário possuir muitos dispositivos de uma maneira ágil, foi feita uma análise mais rigorosa de escalabilidade, através da qual foram levantados os seguintes pontos:

1) A identificação do usuário, por mais que o possibilite ter mais de um, ainda é dada através do UDID (*Unique Device ID*). Tal prática para identificação de dispositivo é depreciada e já deixa de ser possível nas versões mais recentes [2013/2014] do Software Development Kit do iOS. Ou seja, em pouco tempo se tornaria impossível obter esse identificador;
2) Apesar de incomum, é desejável que um usuário possua mais de uma conta de um aplicativo em um celular, o que se torna impossível através da estrutura proposta anteriormente.

Tendo estas questões em vista, é preciso encontrar um novo identificador único para o usuário que seja fácil de obter e que identifique cada usuário individualmente, em vez identificar o dispositivo.

Dentre todas as possibilidades de solução deste problema, foram selecionadas para análise as duas que melhor se encaixavam no problema, levando em consideração a facilidade de implementação. São elas:

a) Criar um sistema de login nativo: sendo este o approach mais clássico, criar um sistema de login próprio seria uma alternativa para criar identificadores únicos para os usuários, de modo que cada um usaria

seu login e senha para acessar o app. Esta é uma solução fácil do ponto de vista do back-end – apenas a criação de campos de Nome de Usuário e Senha na entidade User, bem como a criação de views para requisições de registro de usuário e login;

b) Criar um sistema de login integrado com o Facebook: sendo este um approach mais moderno, o aplicativo requer ao usuário que este libere a integração de sua conta do Facebook com o aplicativo (ciente das informações que serão compartilhadas com o app). Após liberada a conexão com a conta do Facebook, o programa tem acesso à lista de amigos do usuário, seu Facebook ID, bem como o direito de postar na *timeline* do usuário. A implementação desse tipo de login é mais complexa do lado do front-end, mas conta com o suporte da SDK do Facebook, que é bem desenvolvida e documentada.

Depois de definidas as duas possiblidades, decidiu-se por optar pelo login do Facebook, já que este último provê uma série de facilidades, tanto para o usuário quanto para o desenvolvimento do programa:

- O usuário não precisa digitar nenhuma informação, nem escolher uma senha de acesso. Ele precisa, somente uma vez, autorizar o acesso do aplicativo a sua conta do Facebook. O login do usuário pode, então, ser feito pressionando o botão 'Login com o Facebook' toda vez que a sessão for encerrada;
- O aplicativo, tem, então, acesso a toda a lista de amigos do Facebook do usuário, bem como seus nomes e fotos. Com posse destes dados, tornar o Dyfocus uma rede social com possibilidade de adicionar amigos e compartilhar FOFs se torna um trabalho consideravelmente mais simples – e a viralização (capacidade de expansão rápida e simples) é uma das características fundamentais dos apps de maior sucesso dos dias de hoje.

A única grande desvantagem desta abordagem é que o usuário, para ter acesso às funcionalidades do app, tem que ter uma conta do Facebook. No entanto, considerando o crescimento do Facebook e levando em conta que o público alvo do app são jovens portadores de iPhones, a chance de um usuário não ter registro no FB é ínfima.

Decidida a abordagem, tem-se o novo identificador único do usuário: o *facebook_id*, que pode ser obtido quando o usuário libera o acesso do app a sua conta no Facebook.

Observa-se, ainda, que é possível associar um nome e um email ao usuário, que podem ser obtidos através da sua conta na rede social.

A tabela do usuário no banco de dados se torna:

```
User (id[pk], name, email, facebook_id, pub_date);
```

Decide-se por manter a relação **Device** para que haja compatibilidade com os dispositivos que tinham a versão anterior do aplicativo. O usuário seria obrigado a fazer login com o Facebook de qualquer maneira, mas, caso seu UDID constasse no banco de dados, sua conta do Facebook seria associada a sua instância já existente no banco de dados *[quando esta mudança foi proposta, o aplicativo já contava com approx. 30 usuários cadastrados e uma versão na App Store, de modo que descartar a base de dados significaria a perda do acesso de vários usuários a suas fotos].*

Uma vez que mudanças estruturais seriam feitas e o login com Facebook possibilitaria criar todo o aparato social, decidiu-se por, também, criar a estrutura de banco de dados para dar suporte a esse aparato.

É necessário criar, então, uma nova entidade no banco de dados:

- Modelo **Friends**: modelo para representar uma relação de amizade entre duas instâncias de usuários no banco de dados;

Este modelo contará, basicamente, com a referência de dois usuários que são conectados por uma relação de amizade. Sua tabela no banco de dados será:

```
Friends (id[pk], user_1_id[fk], user_2_id[fk]);
```

Depois de mapeada a relação de amizade entre dois usuários, o modelo da base de dados se torna mais complexo, conforme a Figura 6.

Pode-se observar que a estrutura já conta com vários mapeamentos relacionais complexos, como a obtenção de todas as Frames das FOFs de um usuário que tem relação de amizade com outro'.

O modelo apresentado nesta seção foi utilizado no servidor do Dyfocus durante muito tempo, até que se concluiu, através do feedback de muitos usuários, que uma das premissas das quais ele partiu era falha: nem todo usuário está disposto a associar sua conta do *Facebook* a um aplicativo desconhecido.

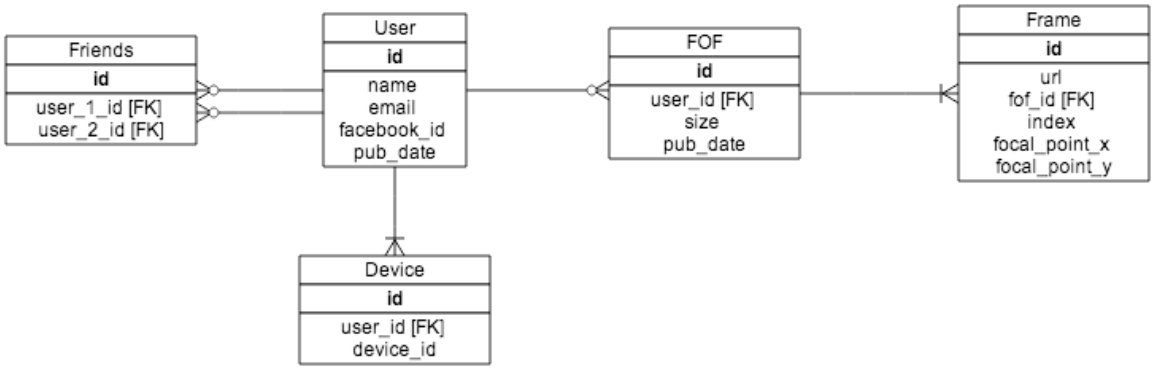

Figura 6 - Incorporação da relação Friend ao banco de dados;

A interface Python com tal estrutura de banco de dados se torna:

```
1. import datetime
2. from django.utils import timezone
3. from django.db import models
4.
5. class User(models.Model):
6.     name = models.CharField(max_length=50, null=True)
7.     device_id = models.CharField(max_length=200, null=True)
8.     email = models.CharField(max_length=200, null=True)
9.     facebook_id = models.CharField(max_length=100, null=True)
10.    pub_date = models.DateTimeField('date published')
11.
12.    def __unicode__(self):
13.        return self.name
14.
15. class Friends(models.Model):
16.    friend_1 = models.ForeignKey(User, related_name='user_friend_1')
17.    friend_2 = models.ForeignKey(User, related_name='user_friend_2')
18.
19. class Device(models.Model):
20.    user = models.ForeignKey(User)
21.    device_id = models.CharField(max_length=200)
22.
23. class FOF(models.Model):
24.    user = models.ForeignKey(User)
25.    size = models.IntegerField()
26.    pub_date = models.DateTimeField('date published')
27.
28.    def __unicode__(self):
29.        return self.name
30.
31. class Frame(models.Model):
32.    url = models.CharField(max_length=200)
```

```
33.     fof = models.ForeignKey(FOF)
34.     index = models.IntegerField()
35.     focal_point_x = models.IntegerField()
36.     focal_point_y = models.IntegerField()
37.
38.     def __unicode__(self):
39.         return self.url
```

4.3.4: Análise Avançada – Iteração Final

Após receber centenas de mensagens de usuários com reclamações de que era impossível utilizar o aplicativo sem a associação com uma conta do Facebook, tornou-se inevitável elaborar um sistema de login nativo e, consequentemente, uma estrutura de banco de dados que desse suporte a ele.

Para isso, foi adicionado ao modelo **User** um campo *password*, que será necessário para que o usuário acesse o aplicativo caso não tenha uma conta de Facebook associada.

A entidade **User** se torna, então:

User (id[pk], name, email, password, facebook_id, pub_date);

Apesar de parecer a menor das iterações do ponto de vista do MySQL, esta alteração foi a com maiores implicações na interace entre front-end e back-end. Isto se deve ao fato de que, até o momento desta mudança, a informação utilizada para identificar o usuário no servidor era seu *facebook_id*. A partir do momento que usuários sem tal identificação foram introduzidos no banco de dados, foi necessário criar uma interface que os contemplasse.

Para tal fim, antes de que o usuário fizesse a requisição de login (contemplada no Capítulo 6), ele necessitaria do *id* do usuário no banco de dados. De modo a evitar uma transação a mais, durante o login, o front-end envia dois possíveis indicadores: o nome do usuário (único, digitado no formulário de login) OU um *facebook_id*, obtido através da API do facebook integrada no app. De posse dessas duas informações, o servidor é capaz de identificar unicamente um usuário e enviar uma resposta com suas informações para o aplicativo, bem como o seu *id*, de fato, que será armazenado no app para todas as requisições que não sejam de *login*.

4.3.5: Adição de Features de Redes Sociais

Definida a estrutura base final do app, foram criadas *features* adicionais, de modo a dar uma aparência de rede social ao aplicativo: **Featured FOFs**, **Likes** e **Comments**. Por não serem fundamentais, será apresentada aqui apenas a estrutura relacional final da base de dados, contemplando estas *features* extras, conforme a Figura 7.

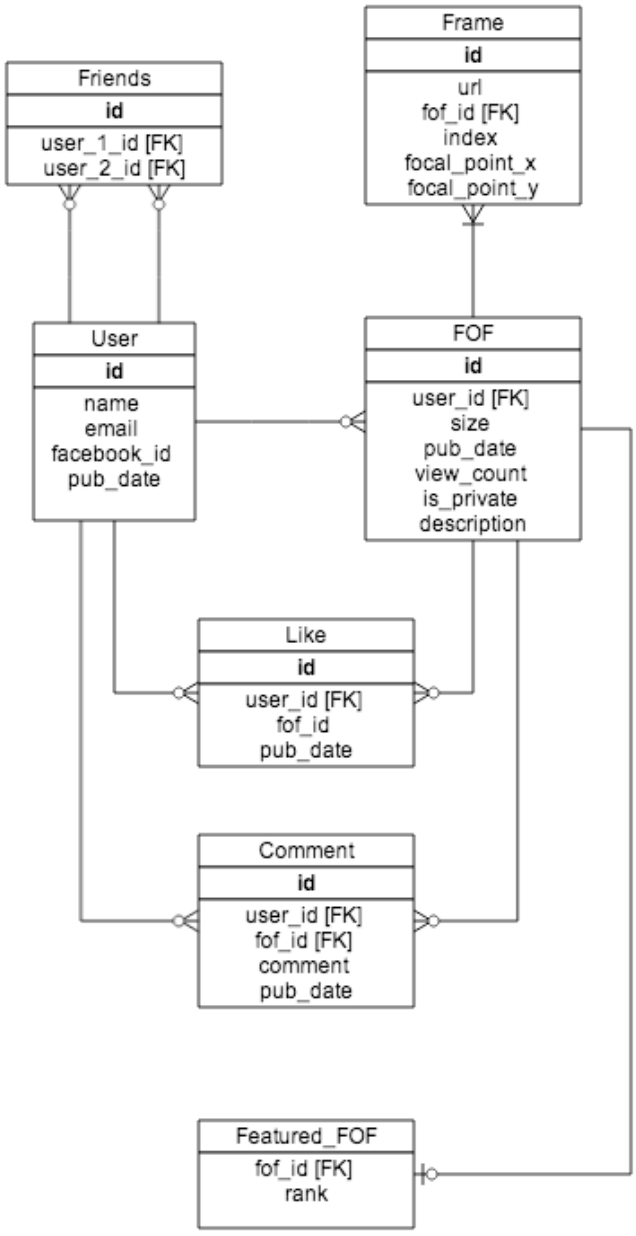

Figura 7 - Estrutura final da base de dados com Features de Redes Sociais

Capítulo 5: Implementação do Back-End

Neste capítulo, serão descritas a definição e a implementação de algumas das principais funcionalidades do servidor, de modo a prover suporte para o front-end do Dyfocus. Adotando uma abordagem *top-down* [51], faz-se necessário entender o fluxo de informações transmitido ao usuário, para, então, entender quais funcionalidades serão necessárias.

Assim como no capítulo anterior, todo o trabalho aqui descrito foi desenvolvido, em sua íntegra, pelo autor da monografia.

5.1: Apresentação do Fluxo de Informações do Aplicativo (front-end)

O fluxo de informações no front-end do Dyfocus é organizado conforme apresentado na Figura 8. Será feita uma breve explicação do significado de cada um dos estados apresentados no diagrama, de modo que se compreenda melhor quais as funcionalidades necessárias em cada um deles.

- **Estado 0:**

  O aplicativo se inicia na tela de login (estado 0), na qual o usuário poderá se cadastrar como um novo usuário – através de email ou de uma conta no Facebook – ou, logar com um login e senha válidos já registrados no banco de dados do servidor.

  Após a validação do login, o fluxo segue para algum dos estados numerados de 1 a 5.

- **Estado 1:**

  O objetivo deste estado é mostrar as fotos dinâmicas públicas dos os usuários da rede, tendo a opção de filtrá-las por "*Top Rated*" (estado 1.1) ou "*Trending*" (estado 1.2). Por default são mostradas as fotos *Top Rated* pois estas são mais atrativas para novos usuários.

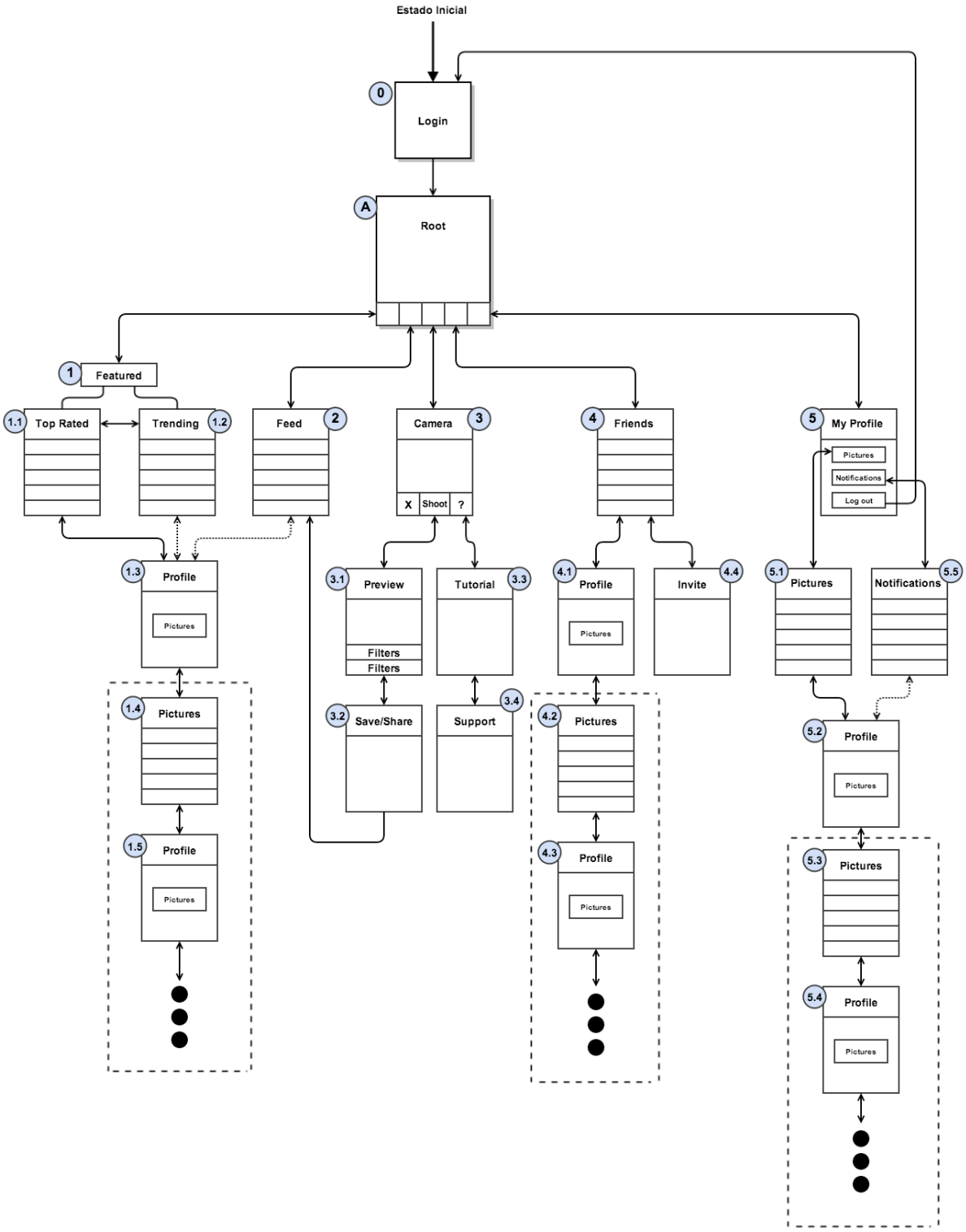

Figura 8 - Fluxo de informações no Front-End do Dyfocus

O estado 1.1 mostra as fotos mais bem-“rankeadas” na rede, ou seja, as fotos de melhor qualidade ou que agradaram mais aos usuários. O

rank pode ser calculado a partir do número de curtidas e comentários de cada foto. Dessa maneira as fotos mais interessantes serão mostradas em posições mais superiores.

O estado 1.2 (*Trending*) é responsável por mostrar as fotos publicadas por todos os usuários na rede em ordem cronológica (das mais novas paras as mais antigas), ou seja, toda vez que qualquer usuário tira uma foto dinâmica e salva ela como pública esta foto é posta no topo da pilha do *Trending* e fica disponível para todas as pessoas na rede do aplicativo. O fluxo a partir desta view é análogo ao do estado 1.1, a representação é feita pela seta pontilhada na figura 3.

- **Estado 2:**

  A seção do *Feed* (estado 2) é análoga as seções 1.1 e 1.2, com a peculiaridade de mostrar apenas as fotos dos seus amigos na rede ou pessoas que você decidiu seguir (*follow*). O fluxo também é análogo ao do estado 1.1, por isso a representação pela seta pontilhada no diagrama.

- **Estado 3:**

  Neste estado, o usuário poderá escolher múltiplos pontos focais em uma cena qualquer através do toque na tela, onde cada toque registrará um ponto focal diferente a ser tomado em consideração pelo aplicativo e processado pelos seus controladores (capítulo 4.2).

  Após selecionados os pontos focais e pressionado o botão de *shoot* (capturar) o usuário será levado ao estado 3.1 (*Preview*). Nesse estado será possível adicionar efeitos de cores e constrastes diferentes em cada imagem de acordo com seu respectivo ponto focal, gerando os mais variados e interessantes resultados finais nas fotos dinâmicas.

  A partir do estado 3.1 é possível retornar ao estado anterior para uma nova tomada de fotos ou seguir o fluxo e alcançar o estado 3.2 onde serão oferecidas as opções de compartilhar a foto publicamente (na rede do aplicativo ou em redes sociais externas como o facebook) ou apenas salvar na galeria de fotos particular do usuário.

Após finalizado o estado 3.2, ou seja, após o usuário ter salvo ou compartilhado a foto dinâmica, o usuário será redirecionado automaticamente para o seu *Feed* onde poderá observar a sua foto recém tirada ou a dos seus amigos.

- **Estado 4:**

  Na view *Friends* serão listados ao usuário todos os seus amigos, dando ao usuário a opção de checar a galeria de fotos de qualquer uma dessas pessoas que estejam na sua lista de amigos/pessoas seguidas. Além disto, também serão listados todos os seus amigos do Facebook que ainda não possuam o aplicativo instalado no celular, possibilitando, assim, o envio de convites (*invites*) a essas pessoas para que elas também passem a utilizar o aplicativo, objetivando assim a disseminação do produto de maneira "viral". Essa função é representada pela view 4.4.

- **Estado 5:**

  Nesta seção, o usuário poderá navegar sobre seu próprio perfil. O usuário poderá checar seu número amigos, além de poder averiguar toda a sua galeria de fotos e notificações que tenha recebido (curtidas e comentários). A galeria de fotos está sendo representada pelo estado 5.1 no diagrama, e a lista de notificações pelo estado 5.5 (*Notifications*). Através da lista de notificações o usuário também poderá acessar o perfil das pessoas responsáveis por tais notificações, gerando um fluxo análogo ao do estado 5.1, por isso a representação por linha pontilhada no diagrama da figura 3.

  A partir do estado 5, o usuário poderá também deslogar da rede do aplicativo, retornando assim ao estado inicial (*Login*) onde poderá ser feito um novo login com um usuário diferente – recomeçando, assim, todo o fluxo do aplicativo.

5.2: Definição das requisições necessárias para cada um dos estados do aplicativo

Considerando as informações que a interface do necessita mostrar para o usuário em cada um dos estados, são levantadas possíveis requisições web para cada um deles. É importante notar que esse é um levantamento geral, e que pode haver redundâncias que serão discutidas no momento da implementação *de facto*. Segue o levantamento na Tabela 2:

Estado	Requisição	Método	Argumentos	Resultado Esperado
0	login	POST	username ou facebook_id	JSON com confirmação de login e user_id correspondente
1	fetch_featured_fofs	GET	-	JSON com lista de FOFs e informações de seus autores
1	fetch_trending_fofs	GET	-	JSON com lista de FOFs e informações de seus autores
2	fetch_user_feed	POST	user_id	JSON com lista de FOFs e informações de seus autores
3	upload_image	POST	user_id, image, frame_index, fof_size, fof_name	JSON com resultado do upload (sucess ou error)
4	fetch_user_friends	POST	user_id	JSON com lista de amigos do usuário
4 e 5	fetch_user_profle	POST	user_id	JSON com informações do usuário

5	logout	POST	user_id	JSON com confirmação de logout

Tabela 2 - Levantamento preliminar de requisições necessárias para o aplicativo Dyfocus

A partir deste levantamento preliminar, já é possível iniciar a implementação das funções básicas do servidor, bem como suas interações com o banco de dados, o que é feito de maneira quase transparente através do Django.

No entanto, antes da codificação, é preciso levar em consideração alguns pontos:

- Fazer várias requisições é mais custoso do que fazer uma única grande (i.e., que retorna uma resposta maior), uma vez que cada requisição traz consigo intrinsecamente o tempo de latência da troca de mensagens;
- A atomização das requisições, se sincronamente vinculada com a chegada a cada um dos estados do app, implica que cada página, antes de ser capaz de apresentar a informação desejada, deve apresentar uma tela intermediária com algum indicador de que os dados estão sendo carregados (requisitados);
- Algumas ações podem ser tratadas no front-end, sem qualquer informação para o servidor, uma vez que elas não remetem a nenhuma ação interna do back-end. A requisição de *logout*, por exemplo, é desnecessária neste contexto, uma vez que o servidor não tem qualquer controle de sessão do usuário.

Com isso em mente, algumas modificações são propostas:

- ✓ Criar um buffer prévio de informações para a abertura do app, concentrando na requisição de *login* as requisições *fetch_featured_fofs*, *fetch_trending_fofs* e *fetch_user_friends*;
- ✓ Eliminar a requisição desnecessária de *logout*.

É importante salientar que as requisições que serão agregadas à de login não deixarão de existir isoladamente – elas ainda poderão ser necessárias em outros

casos, como quando o usuário já tem o aplicativo aberto (i.e., já fez login), mas deseja atualizar as informações sem fechar o app.

Com as modificações propostas, tem-se as informações necessárias para implementação das *views* do Django, que estão sumarizadas na Tabela 3.

Identificador	Requisição	Método	Argumentos	Resultado Esperado
REQ1	login	POST	username ou facebook_id, user_fb_friends	JSON com confirmação de login, user_id correspondente, lista de featured FOFs, trending FOFs e amigos do usuário
REQ2	fetch_featured_fofs	GET	-	JSON com lista de FOFs e informações de seus autores
REQ3	fetch_trending_fofs	GET	-	JSON com lista de FOFs e informações de seus autores
REQ4	fetch_user_feed	POST	user_id	JSON com lista de FOFs e informações de seus autores
REQ5	upload_image	POST	user_id, image, frame_index, fof_size, fof_name	JSON com resultado do upload (sucess ou error)
REQ6	fetch_user_friends	POST	user_id	JSON com lista de amigos do usuário
REQ7	fetch_user_profle	POST	user_id	JSON com

			informações do usuário

Tabela 3 - Lista final de requisições HTTP do back-end do Dyfocus

De posse da lista final das requisições necessárias para o aplicativo, pode-se iniciar a implementação.

5.3: Implementação das requisições

Nesta seção, serão selecionadas algumas requisições listadas na Tabela 3 para a demonstração de sua implementação.

5.3.1: REQ1 – Login

Sendo a requisição inicial e mais completa em termos de dados recuperados, o sistema de login é implementado da seguinte forma:

5.3.1.1: Arquivo *urls.py*

No arquivo urls.py, consta a seguinte expressão regular referente à URL de requisição de login:

```
1.  url(r'^login/$', 'login'),
```

5.3.1.2: Arquivo views.py

No arquivo de *views*, é feita a implementação do login, utilizando modelos de database como **User**, **FOF**, **Friend**, dentre outros. Segue o código Python:

```
1.  @csrf_exempt
2.  def login(request):
3.      # Converte post em JSON para dicionario:
4.      json_request = json.loads(request.POST['json'])
5.
6.      user_name = json_request['name']
7.      user_email = json_request['email']
8.
9.      # Carrega lista de dicionarios com replies:
10.     response_data = {}
11.
12.     try:
13.         user_facebook_id = json_request['facebook_id']
```

```
    except:
        user_facebook_id = False
        user_password = json_request['user_password']

    # Trata o usuario, caso ele nao tenha feito login com Facebook:
    if not user_facebook_id:
        try:
            user = User.objects.get(email=user_email)
        except (KeyError, User.DoesNotExist):
            user = User(name = user_name, pub_date=timezone.now(), email = user_email)
        else:
            # Confere se a senha esta correta:
            if user.password != user_password:
                response_data['error'] = "Email/Password doesn't match"
                return HttpResponse(json.dumps(response_data), mimetype="application/json")

    # Trata o usuario, caso ele tenha feito login com Facebook:
    else:
        try:
            user = User.objects.get(facebook_id=user_facebook_id)
        except (KeyError, User.DoesNotExist):
            user = User(dname = user_name, facebook_id = user_facebook_id, pub_date=timezone.now(), email = user_email)


    # Ja temos um objeto user, agora vamos coletar/atualizar suas informacoes:

    response_data['user_id'] = user.id
    response_data['user_friend_count'] = len(User.objetcs.filter(friend_1_id=user.id)) + len(User.objetcs.filter(friend_2_id=user.id))

    # Caso o usuario venha do Facebook, ele envia para essa requisicaoo uma lista de amigos:
    try:
        user_friend_list = json_request['friends']
    except:
        response_data['friends_list'] = user.fetch_friend_list()
    else:
        response_data['friends_list'] = user.update_friend_list(user_friend_List)

    # Lista de Featured FOFs:
    response_data['featured_fof_list'] = [dict(
        frames = [dict(frame_url=frame.url,frame_index=frame.index) for frame in featured_fof.fof.frame_set.all().order_by('index')[:5]],
        user_name = featured_fof.fof.user.name,
        user_id = featured_fof.fof.user.id,
        user_facebook_id = featured_fof.fof.user.facebook_id,
        id = featured_fof.fof.id,
        is_private = featured_fof.fof.is_private,
        fof_name = featured_fof.fof.name,
        pub_date = featured_fof.fof.pub_date,
        fof_description = featured_fof.fof.description,
        comments_count = len(featured_fof.fof.comment_set.all()),
        likes_count = len(featured_fof.fof.like_set.all()),
    ) for featured_fof in Featured_FOF.objects.all().order_by('-rank')]
```

```
65.
66.     # Lista de Trending FOFs:
67.     response_data['trending_fof_list'] = [dict(
68.         frames = [dict(frame_url=frame.url,frame_index=frame.index) for fra
    me in fof.frame_set.all().order_by('index')[:5]],
69.         user_name = fof.user.name,
70.         user_id = fof.user.id,
71.         user_facebook_id = fof.user.facebook_id,
72.         id = fof.id,
73.         is_private = fof.is_private,
74.         fof_name = fof.name,
75.         pub_date = fof.pub_date,
76.         fof_description = fof.description,
77.         comments_count = len(fof.comment_set.all()),
78.         likes_count = len(fof.like_set.all()),
79.     ) for fof in FOF.objects.all().filter(is_private=False).order_by('-
    pub_date')[:50]]
80.
81.
82.     return HttpResponse(json.dumps(response_data), mimetype="aplication/jso
    n")
```

5.3.2: REQ5 – Upload de Imagem

Para a requisição de upload de FOF, são passados como parâmetro POST o nome da FOF associada, o tamanho (número de frames) que a FOF possui, bem como o id do usuário que está fazendo o upload (autor da FOF).

É importante notar que todas as frames são enviadas em um mesmo POST request, e cada um dos arquivos correspondentes é identificado por uma *key*, que é usado para recuperá-los na *view* de upload.

5.3.2.1: Arquivo *urls.py*

No arquivo urls.py, consta a seguinte expressão regular referente à URL de requisição de login:

```
2.  url(r'^upload_image/$', 'upload_image'),
```

5.3.2.2: Arquivo views.py

Segue a implementação da requisição de upload de imagem para o servidor S3:

```
1. @csrf_exempt
```

```
2. def upload_image(request):
3.
4.     # Recupera as informacoes POST da FOF:
5.     fof_size = request.POST['fof_size']
6.     fof_name = request.POST['fof_name']
7.     user_id = request.POST['user_id']
8.
9.     response_data = {}
10.
11.    # Procura o usuario correspondente:
12.    try:
13.
14.        user = User.objects.get(id=user_id)
15.
16.        # Procura a FOF ou cria uma:
17.        try:
18.            frame_FOF = FOF.objects.get(name=fof_name)
19.        except (KeyError, FOF.DoesNotExist):
20.            frame_FOF = user.fof_set.create(name = fof_name, size = fof_siz
    e, pub_date=timezone.now(), view_count = 0, is_private = 0)
21.
22.        #Conecta ao S3 com a AWS_ACCESS_KEY_ID e a AWS_SECRET_ACCESS_KEY:
23.        conn = S3Connection('xxxxxxxxxxxx-access-key-id', 'xxxxxxxxxxxxxx-
    access-key-secret')
24.        b = conn.get_bucket('dyfocus')
25.
26.        #Cria todas as frames de uma vez:
27.        for i in range(int(fof_size)):
28.
29.            # Key da imagem atual, baseado no index da iteracao:
30.            key = 'apiupload_' + str(i)
31.
32.            image = Image.open(cStringIO.StringIO(request.FILES[key].read()
    ))
33.
34.            out_image = cStringIO.StringIO()
35.            image.save(out_image, 'jpeg')
36.
37.            frame_focal_point_x_key = 'frame_focal_point_x_' + str(i)
38.            frame_focal_point_y_key = 'frame_focal_point_y_' + str(i)
39.
40.            # Obtem a posicao do ponto focal para esta frame
41.            frame_focal_point_x = request.POST[frame_focal_point_x_key]
42.            frame_focal_point_y = request.POST[frame_focal_point_y_key]
43.
44.            # Cria um identificador unico para imagem no seguinte formato:

45.            #frame_name = <user_id>_<fof_name>_<frame_index>.jpeg
46.            frame_name = user_id
47.            frame_name += '_'
48.            frame_name += fof_name
49.            frame_name += '_'
50.            frame_name += str(i)
51.            frame_name += '.jpeg'
52.
53.            # Associa a url correspondente, seguindo o padrao da AWS:
54.            # frame_url = <s3_url>/<frame_name>
55.            frame_url = 'http://s3.amazonaws.com/dyfocus/'
56.            frame_url += frame_name
```

```
57.
58.             frame = frame_FOF.frame_set.create(url = frame_url, index = str
    (i), focal_point_x = frame_focal_point_x, focal_point_y = frame_focal_point
    _y)
59.
60.             k = b.new_key(frame_name)
61.
62.             k.set_contents_from_string(out_image.getvalue())
63.
64.             response_data["result"] = "ok"
65.
66.     except (KeyError, User.DoesNotExist):
67.         response_data["error"] = "User does not exist."
68.
69.
70.     return HttpResponse(json.dumps(response_data), mimetype="aplication/jso
    n")
```

5.4: Testes e *Deployment*

Uma vez definidas e implementadas as novas requisições, elas eram testadas localmente (alguns exemplos de respostas podem ser vistos na seção subsequente). Para tanto, criava-se o *Front-End* que necessitaria da resposta dessa requisição, alterava-se a URL do aplicativo de *produção* para *teste* – i.e., ele apontaria seus *requests* para um ip local em vez de para o servidor *dyfoc.us*.

Após toda essa implementação, as novas funcionalidades eram testadas exaustivamente no App, com a intenção de gerar todos os tipos de *bug* possíveis, de modo a corrigi-los.

Assim que a equipe não conseguia mais gerar erros (passíveis de serem tratados no back-end), concluía-se que o servidor já podia ter seu código fonte atualizado.

Para isso, criava-se um *branch* novo na ferramenta de versionamento (Git), para o qual se fazia um commit da modificação já testada e aprovada. Então, essa nova versão era baixada na produção (através de acesso *SSH – Secure Shell*) e o servidor Apache era reiniciado.

Sempre que esse procedimento era repetido, era necessário ficar em plantão para que qualquer correção pudesse ser prontamente feita caso algum bug instituído pudesse, por algum motivo, tirar o serviço do ar.

5.5: Respostas das Requisições

São mostradas, aqui, algumas respostas obtidas ao realizar requisições remetentes àquelas apresentadas anteriormente neste trabalho.

5.5.1: Exemplo de resposta da requisição REQ1:

Quando feito um post na URL /login/ com as informações necessárias, a resposta obtida tem o seguinte formato:


```
{
    "user_friend_count": 58,
    "user_id": 38,
    "featured_fof_list": [
        {
            "liked": "0",
            "user_id": 591,
            "pub_date": "05/09/2013",
            "comments": 2,
            "fof_name": "57447.010765",
            "user_facebook_id": "100001650257978",
            "fof_description": "",
            "frames": [
                {
                    "frame_url": "http://s3.amazonaws.com/dyfocus/100001650257978_57447.010765_0.jpeg",
                    "frame_index": 0
                },
                {
                    "frame_url": "http://s3.amazonaws.com/dyfocus/100001650257978_57447.010765_1.jpeg",
                    "frame_index": 1
                }
            ],
            "user_name": "RaÃssa Valletta",
            "id": 1471,
            "is_private": 0,
            "likes": 80
        },
        {
            "liked": "0",
            "user_id": 1754,
            "pub_date": "06/05/2013",
            "comments": 1,
            "fof_name": "146852.374163",
            "user_facebook_id": "839260260",
            "fof_description": "",
            "frames": [
                {
                    "frame_url": "http://s3.amazonaws.com/dyfocus/839260260_146852.374163_0.jpeg",
                    "frame_index": 0
                },
                {
```

```
                    "frame_url": "http://s3.amazonaws.com/dyfocus/839260260_146852.374163_1.jpeg",
                    "frame_index": 1
                }
            ],
            "user_name": "Lee A Dunn",
            "id": 1827,
            "is_private": 0,
            "likes": 57
        },
        {
            "liked": "0",
            "user_id": 2200,
            "pub_date": "06/20/2013",
            "comments": 1,
            "fof_name": "423051.073409",
            "user_facebook_id": "700091928",
            "fof_description": "",
            "frames": [
                {
                    "frame_url": "http://s3.amazonaws.com/dyfocus/2200_423051.073409_0.jpeg",
                    "frame_index": 0
                },
                {
                    "frame_url": "http://s3.amazonaws.com/dyfocus/2200_423051.073409_1.jpeg",
                    "frame_index": 1
                }
            ],
            "user_name": "Zsuiram Ikcejyrts",
            "id": 2158,
            "is_private": 0,
            "likes": 24
        },
    ],
    "friends_list": [
        {
            "name": "Heithor Zanini",
            "friend_count": 1,
            "id_origin": 0,
            "friends": 4,
            "facebook_id": "1446722246",
            "id": 69
        },
        {
            "name": "CÃ¡ssio Marcos Goulart",
            "friend_count": 46,
            "id_origin": 0,
            "followers_count": 32,
            "facebook_id": "100000491430286",
            "id": 74
        },
        {
            "name": "Luciano Araujo Do Nascimento",
            "friend_count": 30,
            "id_origin": 0,
            "followers_count": 25,
            "facebook_id": "100002082427412",
```

```
            "id": 85
        },
            {
                "name": "Pedro Kurtz",
                "friend_count": 0,
                "id_origin": 0,
                "followers_count": 12,
                "facebook_id": "797653776",
                "id": 92
            }
        ],

        "trending_fof_list": [
            {
                "liked": "0",
                "user_id": 73,
                "pub_date": "11/05/2013",
                "comments": 0,
                "fof_name": "164633.578858",
                "user_facebook_id": "100001077656862",
                "likes": 1,
                "frames": [
                    {
                        "frame_url": "http://s3.amazonaws.com/dyfocus/73_164633.578858_0.jpeg",
                        "frame_index": 0
                    },
                    {
                        "frame_url": "http://s3.amazonaws.com/dyfocus/73_164633.578858_1.jpeg",
                        "frame_index": 1
                    }
                ],
                "user_name": "Marcelo Salloum Dos Santos",
                "id": 3371,
                "is_private": 0,
                "fof_description": "Octopus"
            },
            {
                "liked": "0",
                "user_id": 74,
                "pub_date": "11/05/2013",
                "comments": 0,
                "fof_name": "74136.262971",
                "user_facebook_id": "100000491430286",
                "likes": 0,
                "frames": [
                    {
                        "frame_url": "http://s3.amazonaws.com/dyfocus/74_74136.262971_0.jpeg",
                        "frame_index": 0
                    },
                    {
                        "frame_url": "http://s3.amazonaws.com/dyfocus/74_74136.262971_1.jpeg",
                        "frame_index": 1
                    }
                ],
                "user_name": "CÃ¡ssio Marcos Goulart",
```

```
                    "id": 3370,
                    "is_private": 0,
                    "fof_description": "Olha\nO\nQueee\neeeeee\nijo!"
                },
                {
                    "liked": "0",
                    "user_id": 654,
                    "pub_date": "10/26/2013",
                    "comments": 0,
                    "fof_name": "33128.438315",
                    "user_facebook_id": "1161056267",
                    "likes": 4,
                    "frames": [
                        {
                            "frame_url": "http://s3.amazonaws.com/dyfocus/654_33128.438315_0.jpeg",
                            "frame_index": 0
                        },
                        {
                            "frame_url": "http://s3.amazonaws.com/dyfocus/654_33128.438315_1.jpeg",
                            "frame_index": 1
                        }
                    ],
                    "user_name": "Isanulfo Cordeiro",
                    "id": 3317,
                    "is_private": 0,
                    "fof_description": "On my mind, nothing but the taste of a good ol' scotch."
                }
            ]
        }
```


5.5.2: Exemplo de resposta da requisição REQ5:

Após realizada uma requisição POST de upload, caso haja sucesso, o seguinte resultado é obtido:


```
{
    "result": "ok"
}
```

Capítulo 6: Resultados

6.1: Considerações Preliminares

O Dyfocus foi disponibilizado ao público no dia 10 de outubro de 2012, e, desde então, já apresentou onze atualizações oficiais. Antes de qualquer divulgação, já alcançou a marca de mil usuários em menos de um mês.

Vários melhoramentos foram feitos desde sua concepção do ponto de vista da UX (*user experience*), que tiveram reflexos evidentes em como os usuários utilizavam o aplicativo. Foi possível perceber quais foram as mudanças efetivas e quais foram aparentemente inúteis no sentido de melhorar a compreensão da funcionalidade/finalidade do aplicativo.

Estas medidas foram orientadas, também, pelo *feedback* dos usuários, que tinham acesso ao e-mail e suporte do aplicativo para dar sugestões, opiniões e fazer críticas ou reclamações.

Foi possível perceber que o crescimento no número de usuários é mais difícil do que o esperado, e que atingir uma meta viral é uma tarefa árdua e que conta com muito mais variáveis do que o pensado no início do projeto.

Ao final de um ano, o aplicativo conseguiu atingir um número de usuários próximo a 9 mil, o que, apesar de significativo, ainda é bem aquém do necessário para ser considerado um "*app de sucesso"*.

O Dyfocus se tornou um grande laboratório da Cheesecake Labs, através do qual pudemos (e ainda podemos) fazer experimentos, analisar implicações de modificações, testar *Google Ads* e coisas do tipo. Serviu como ferramenta de integração da equipe (que hoje conta com mais de 10 pessoas) e como ponto de partida para projetos atuais, hoje muito mais bem-sucedidos e bem-estruturados.

Foram tentadas várias maneiras de *monetizar* o *app*, sendo uma delas a implantação de anúncios (Google Ad Mobile), que não gerou resultados significativos. Foi também feita uma versão *ad-free*, que nunca chegou a ser publicada.

Atualmente, o aplicativo ainda se encontra disponível para download gratuito na *App Store*, mas sua manutenção é feita, hoje, de maneira pontual e esporádica. No entanto, não existe intenção de descontinuar a distribuição do app e/ou desligar seus servidores.

6.2: Estatísticas

Para medição dos resultados obtidos, além das próprias informações guardadas no banco dados, utilizamos também uma ferramenta de medição e análise muito poderosa chamada Flurry Analytics [52]. Com a integração do Flurry no aplicativo Dyfocus, fez se possível rastrear cada passo de cada usuário dentro do *app* através de eventos disparados a cada ação do usuário. Além disso, é possível também rastrear informações geográficas e de localização de cada usuário.

Utilizando os serviços de análise oferecidos pelo Flurry e também nosso próprio banco de dados, após mais de 1 ano de projeto, desenvolvimento, testes e análises, chegamos aos seguintes resultados:

- Número de usuários únicos: 8.832;
- Número de usuários por região:
    - Ásia: 2.732
    - América do Norte: 2.146
    - Europa: 1.853
    - América do Sul: 1.247
    - Oriente Médio: 543
    - Oceania: 143
    - África: 94
    - América Central: 74
- Número de usuários por país predominante:
    - Estados Unidos: 1544;
    - Brasil: 790;

    - Rússia: 694;
    - Tailândia: 333;
    - Reino Unido: 277;
    - China: 270;
    - México: 269;
    - Vietnã: 243;
    - Espanha: 195;
    - Ucrânia: 182;
    - Itália: 177;
    - França: 173;
    - Japão: 165;
    - Alemanha: 157;
    - Índia: 156;
    - Turquia: 148;
    - Canadá: 145;
    - Austrália: 118;
    - Taiwan: 115;
    - Chile: 93;
    - Holanda: 93;
- Número total de imagens armazenadas no servidor S3 da AWS: 7.870;
- Número total de fotos dinâmicas (FOFs) geradas: 4.016;
- Número total de curtidas em FOFs: 1.625;
- Número total de comentários em FOFs: 247;

Visto que as funcionalidades de curtir e comentar foram as últimas implementadas no aplicatico, podemos notar que seus números apresentam-se relativamente inferiores aos demais.

Apresentados os resultados obtidos pelo aplicativo, no próximo capítulo concluiremos sobre a análise dos mesmos e sobre as perspectivas alcançadas, não alcançadas e que ainda temos como meta.

6.3: Possíveis melhorias

Depois de realizados diversos testes, adicionadas ferramentas de estatísticas como Flurry [52] e Google Analytics [53], é possível perceber algumas falhas no projeto que ainda necessitam ser melhoradas para que haja uma maior retenção do usuário:

- As *queries* no banco de dados devem otimizadas, de modo que o usuário não tenha que ficar esperando durante muito tempo as informações serem carregadas no aplicativo;
- A distribuição das informações por requisição deve ser repensada, de modo que cada *request* leve menos tempo e só seja feito no momento em que a informação recuperada através dele se venha a se tornar necessária.

6.4: Considerações Finais

É possível observar que o processo de desenvolvimento de um *back-end* trata de um processo iterativo contínuo e que alcançar uma versão consolidada, confiável e rápida é uma tarefa impraticável em curto ou, mesmo, médio prazo.

A questão da escalabilidade, e o crescimento viral são aspectos que podem ir em conflito direto com toda a estrutura do servidor, uma vez que este, mesmo sendo planejado visando a atender uma enorme demanda, pode apresentar um comportamento completamente diferente do esperado quando o número de usuários cresce a taxas significativas.

Além disso, deve ser levado em consideração que modificações estruturais se tornam cada vez mais complexas à medida que o número de usuários cresce – migrar uma coluna de uma base de dados para outra tabela relacional é uma coisa quando esta tabela tem dez linhas, e outra completamente diferente quando ela tem 50 mil.

O trabalho se torna ainda mais árduo quando problemas de inconsistência surgem, porque alterações no banco de dados não podem ser feitas diretamente naquele ao qual os usuários têm acesso, mas sim numa cópia. Do contrário, o serviço pode ficar indisponível durante horas, o que pode acarretar na insatisfação de muitos usuários – o que não é interessante de maneira alguma, como pudemos observar, na prática, ao longo do desenvolvimento deste projeto.

Lidar com tais inconsistências significa não só modificar o espelho local do banco de dados, mas se assegurar que serão criadas funções de migração para os dados mais recentes que não constam na base de dados local – tarefa que pode se tornar extremamente trabalhosa e, caso seja executada de maneira imprudente, pode acarretar, também, na indisponibilização do serviço.

Capítulo 7: Conclusão

Após a realização deste trabalho, foi possível observar, na prática, diversos aspectos relacionados ao desenvolvimento de aplicativos, criação de empresa, interação com o usuário e planejamento da UX (experiência do usuário).

Foi possível notar que criar um programa de fácil usabilidade e de fácil compreensão de suas capacidades/funcionalidades não é uma tarefa trivial e demanda um conhecimento muito maior do que o imaginado pela grande maioria das empresas que visam a criar um *app* que possa ser denominado “de sucesso”.

Acima de tais fatores, o desenvolvimento de um aplicativo que visa a atender aos *standards* do mercado é uma tarefa que envolve a atenção em todas as etapas – desde a escolha da tecnologia de implantação (linguagem, plataforma, serviços) até o atendimento ao usuário final (sistema de *feedback*, velocidade de resposta).

Para isso, foi fundamental levar em consideração os conhecimentos obtidos no curso, que forneceram um *background* que orientou o autor desta monografia a como encontrar e desenvolver soluções que pudessem ser, de alguma maneira, interessantes e competitivas.

Pode-se dizer, sem grandes hesitações, que o trabalho em um mercado que está tão intrinsecamente conectado à tecnologia envolve um esforço muito grande no sentido de conseguir acompanhar as tendências e inovações que surgem a cada segundo.

Isso trouxe a todos que trabalharam neste projeto o desafio de lidar com tecnologias muitas vezes até então desconhecidas, uma vez que emergiram ou se tornaram populares apenas nos últimos meses, surgidas do esforço de promover melhorias de performance ou de facilidade de desenvolvimento.

Tendo toda esta gama de considerações em mente, é possível dizer que o desafio de se desenvolver um aplicativo estável, usável e interessante foi satisfatoriamente alcançado e que este é, sem dúvidas, um ponto de partida para a criação de outros trabalhos, mais robustos e melhores.

Bibliografia:

[01] A. C. Kay, “The Early History of Smalltalk”, Apple Computer, Setembro de 2007.

[02] A. Singh, “A Brief History of Mac OS X”, Mac OS X Internals, Junho de 2012.

[03] C. Garling, “iPhone Coding Language Now World’s Third Most Popular, Wired, Maio de 2013.

[04] R. Wentk, “Cocoa: Volume 5 of Developer Reference Apple Developer Series”, John Wiley and Sons, 2009.

[05] B. Cox, “The object oriented pre-compiler: programming Smalltalk 80 methods in C Language”, ACM SIGPLAN Notices, New York, 1983.

[06] M. Trent e D. McCormack, “Beginning Mac OS X Programming”. Indianapolis, IN: Wiley Pub., 2005.

[07] Apple Inc., “Dynamic Method Resolution”, Objective-C Runtime Programming Guide, Outubro de 2009.

[08] Apple Inc., “Avoiding Messaging Errors”, The Objective-C Programming Language, Outubro de 2009.

[09] B. J. Cox, “Object Oriented Programming: An Evolutionary Approach”, Addilson Wesley, 1991.

[10] C., P. e J. Nicola, “Object-oriented Programming”. Englewood Cliffs N.J: Prentice Hall, 1993.

[11] M. Pilgrim, “Dive into Python”, Nova Iorque, 2004.

[12] B. Venners, “The Making of Python”, Artima Developer, Janeiro de 2003.

[13] Michele Simionato, “The Python 2.3 Method Resolution Order”, Python Software Foundation, Junho de 2008.

[14] P. J. Eby, “PEP 333 – Python Web Server Gateway Interface v1.0”, Python Software Foundation, Dezembro de 2003.

[15] P. Piotrowski, “Build a Rapid Web Development Enviroment for Python Server Pages and Oracle”, Oracle, Junho de 2008.

[16] N. Hamilton, “The A-Z of Programming Language: Python”, Computerworld, Agosto de 2008.

[17] Apple Inc., “Xcode on the Mac App Store”, Outubro de 2012.

[18] Apple Developer, “Mac Dev Center”, Julho de 2011.

[19] M. Trent e D. McCormack, “Beginning Mac OS X Programming”. Indianapolis, IN: Wiley Pub., 2005.

[20] M. Willmore, “TextMate 1.0.1 Review: A Checkmate for TextMate?”, Maczealots.com, Outubro de 2004.

[21] M. Bell, “TextMate: The Missing Editor for OS X”, Drunkenblog, Novembro de 2004.

[22] J. E. Gray, “TextMate: Power Editing for the Mac”. Lewisville, TX: Pragmatic, 2007.

[23] J. J. Garret, “Ajax: A New Approach to Web Applications”, Fevereiro de 2005.

[24] D. Crockford, “Introducing JSON”, json.org, Maio de 2009.

[25] D. Crockford, “JSON: The Fat-Free Alternative to XML”, Julho de 2009.

[26] M. Holm, “JSON: The JavaScript subset that isn’t”, The timeless repository, Maio de 2011.

[27] Di Giacomo, M., "MySQL: lessons learned on a digital library," Software, IEEE, vol.22, no.3, pp.10,13, Maio-Junho de 2005.

[28] V. Vaswani. “MySQL: The Complete Reference”. New York: McGraw-Hill/Osborne, 2004.

[29] P. DuBois. “MySQL: The Definitive Guide to Using, Programming, and Administering MySQL 4.1 and 5.0”. Indianapolis, IN: Sams Pub., 2005.

[30] R. J. T. Dyer, “MySQL in a Nutshell”. Sebastopol, CA: O'Reilly Media, 2008. Godse, Gautam. Xcode. N.p.: John Wiley & Sons, 2010.

[31] C. Aulds, “Linux Apache Web Server Administration”. San Francisco: Sybex, 2001.

[32] L., B. e P. Laurie, “Apache: The Definitive Guide”. Beijing: O'Reilly, 1999.

[33] K. A. L. Coar,e R. C. Bowen, “Apache Cookbook”. Sebastopol, CA: O'Reilly, 2004.

[34] R. S. Huckman, G. P. Pisano e L. Kind, “Amazon Web Services”. Boston, MA: Harvard Business School, 2008.

[35] "WSGI." WSGI — WSGI.org. N.p., n.d. Web. 23 de Fevereiro de 2014.

[36] “Instagram” – Instagram.com, Fevereiro de 2014.

[37] S. Burbeck, “How to use Model-View-Controller (MVC)”, Março de 1997.

[38] P. Mell; T. Grance, “The NIST Definition of Cloud Computing”, *National Institute of Standards and Technology*, Setembro de 2011.

[39] D. Garcia, “Open Step Specification”. Outubro de 1994.

[40] A. Alexandrescu, “Modern C++ Design: Generic Programming and Design Patterns Applied”. Addison-Wesley, 2001.

[41] S. Schach, “Object-Oriented and Classical Software Engineering”. Seventh Edition, McGraw-Hill, 2006.

[42] “Facebook” – Facebook.com, Fevereiro de 2014.

[43] A. Holovaty, J. Kaplan-Moss e J. Dunck, “The Django Book, 2009.

[44] “EC2Instances” – EC2Instances.info, Fevereiro de 2014.

[45] R. Hertzog, “The Debian Administrator's Handbook”. Lulu Press, 2012.

[46] “June 2013 Web Server Survey”, Netcraft, Netcraft.com.

[47] E. Naramore, “Beginning PHP5, Apache, and MySQL Web Development”. Indianapolis, IN: Wiley, 2005

[48] T. Negrino, Tom e D. Smith, “JavaScript”. Berkeley, CA: Peachpit, 2012.

[49] M. Callaghan, Mark, "MySQL at Facebook". Youtube.com 2010.

[50] "Wikimedia servers — System architecture". Wikimedia Meta-Wiki. Wikimedia Foundation, 2012.

[51] M. D. V. Davies “The Administration of International Organizations: Top down and Bottom up”. Aldershot, Hants, Inglaterra: Ashgate, 2002.

[52] “Flurry Analytics” – Flurry.com, Fevereiro de 2014.

[53] “Google Analytics” – analytics.google.com, Fevereiro de 2014.