Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Curso de Graduação em Engenharia de Teleinformática

Tiago Augusto da Silva Bencardino

Estudo comparativo das plataformas Android e iOS para aplicações RESTful

Fortaleza – Ceará
Fevereiro 2013

Autor:
Tiago Augusto da Silva Bencardino

Orientador:
Prof. Dr. José Marques Soares

Estudo comparativo das plataformas Android e iOS para aplicações RESTful

Monografia de Conclusão de Curso apresentada à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Engenheiro de Teleinformática**.

Fortaleza – Ceará
Fevereiro 2013

Tiago Augusto da Silva Bencardino

Estudo comparativo das plataformas Android e iOS para aplicações RESTful

Esta Monografia foi julgada adequada para a obtenção do diploma de Engenheira do Curso de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Tiago Augusto da Silva Bencardino

Banca Examinadora:

Prof. Dr. José Marques Soares
Orientador

Prof. p1

Prof. p2

Prof. p3

Fortaleza, 29 de janeiro de 2013

Resumo

Um grande número de aplicações *Web* têm, para as plataformas móveis *iOS* e *Android*, uma versão para mobiles, onde as mesmas informações são compartilhadas na núvem. Algumas redes sociais populares, como *Twitter* e *Foursquare*, possuem interfaces de comunicação REST como um serviço para outras aplicações. Este trabalho tem como objetivo fazer um estudo comparativo entre aplicações iOS e Android que utilizam um aplicativo web RESTful. Para realizar o estudo, é criada uma pequena rede social de criação de *quizzes*, utilizando *Ruby on Rails* e um serviço de *SaaS*, como o *Heroku*.

Palavras-chave: iOS, Android, Mobile, Ruby on Rails, REST, RESTFul

Abstract

A great number of web applications has, for mobile platforms *iOS* and *Android*, some version for mobile systems, which same information is shared on cloud. Some popular social networks like *Twitter* and *Foursquare* has communication interfaces REST as a service to other applications. This paper aims to make a comparative study between iOS and Android applications that use a RESTful web application. To conduct the study, it created a small social network to create quizzes using *Ruby on Rails* and a SaaS service like *Heroku*.

Keywords: iOS, Android, Mobile, Ruby on Rails, REST, RESTFul.

Dedico este trabalho a uma galera ai.

... Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir.

Steve Jobs

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

Capítulo 1

Aplicação

Este capítulo visa apresentar o contexto no qual o trabalho realizado está inserido, assim como definir seus objetivos e justificar seus propósitos. Na Seção 1.1, é apresentada uma contextualização para a solução gerada. Os objetivos geral e específicos são indicados na Seção 1.2. Por fim, a Seção 1.3 realiza uma breve descrição da metodologia empregada e a Seção 1.4 descreve o formato no qual esta monografia está organizada.

1.1 Contextualização

A computação em nuvem propõe que recursos de hardware e software sejam entregues como serviço pela internet. Sob essa visão, sistemas como web services podem ser hospedados sobre uma infraestrutura de terceiros, de modo a diminuir custos operacionais. Web service é uma solução utilizada para realizar a comunicação entre sistemas distintos, que podem utilizar diferentes tecnologias de implementação e estar sob as mais diversas plataformas. Dessa forma, aplicações feitas por equipes diferentes, em diferentes contextos, podem se comunicar e executar ações ou consumir recursos. Na prática, a função de um web service é fazer com que os recursos da aplicação do software estejam disponíveis na rede de uma forma normalizada, ou seja, sobre um protocolo onde as duas pontas (cliente e servidor) entendam a mensagem. Existem algumas formas padronizadas de implementar serviços, sendo as mais comuns as soluções em SOAP e REST. REST é um padrão de web service mais comum em computação móvel. A computação móvel pode ser representada como um novo paradigma computacional que permite

que usuários desse ambiente tenham acesso a serviços independentemente de sua localização, podendo inclusive, estar em movimento. `https://portal.fucapi.br/tec/imagens/revistas/ed02_04.pdf`. Originalmente, telefones celulares possuíam apenas a capacidade de realizar a comunicação por voz. Com o passar do tempo, algumas funcionalidades extras foram sendo implementadas e, atualmente, os dispositivos possuem um amplo poder de processamento e comunicação. Tais aparelhos receberam a nomenclatura comercial de "smartphone". Smartphones (ou telefones inteligentes) são celulares com funcionalidades avançadas, que podem ser estendidas através de programas adicionados ao seu sistema operacional. Em sistemas operacionais como iOS e Android, existem centenas de milhares de aplicativos que podem ser baixados em lojas virtuais, como App Store e Google Play, respectivamente. As categorias líderes em número de aplicativos baixados são jogos e sociais. Rede social é uma estrutura composta por pessoas ou organizações conectadas, de modo a partilhar valores e objetivos comuns. Existem diversos tipos de redes, como as de relacionamento (Facebook, Orkut, Twitter), redes profissionais (LinkedIn), redes comunitárias (Cromaz), entre outros. Em redes de relacionamentos como Facebook e Orkut, surgiram alguns jogos que buscam interagir as pessoas, aproveitando características de integração social. Tais jogos recebem a denominação de "social network gaming".

1.2 Objetivos

Esta seção visa descrever os objetivos deste trabalho através de um panorama geral de seus propósitos e da descrição de pontos específicos que devem ser atendidos.

1.2.1 Objetivos Gerais

O principal objetivo desse trabalho é a criação de uma plataforma web, hospedada em um serviço de computação em nuvem, que interaja com duas aplicações mobile (em iOS e em Android), através de um serviço de web service utilizando REST. A aplicação web permitirá que pessoas possam criar questionários e enquetes, os quais denominamos de quizzes, e permitirá que essas pessoas acompanhem as respostas marcadas em um relatório. Por sua vez, as aplicações móveis em iOS e Android permitirão que as pessoas possam baixar os quizzes de seu interesse e jogá-los/responde-los. A depender do modo como cada quiz for criado, será creditada uma pontuação por jogo. Além disso, será creditada uma pontuação

referente a criação de cada quiz e a adição de perguntas.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são enumerados a seguir:

i. Análise das tecnologias existentes para implementação da arquitetura proposta

ii. Modelagem de uma interface gráfica de fácil utilização, para sistemas web e mobile

iii. Implementação do sistema web e hospedagem em um serviço de computação em nuvem

iv. Destacar Ruby on Rails como framework web de grande facilidade, fácil manutenção e rápida interação com mobile

v. Implementação mobile das

vi. Comparação entre algumas funcionalidades e implementações: como são feitas em iOS e Android

1.3 Metodologia utilizada

Primeiramente, uma revisão bibliográfica referente às áreas de Computação em nuvem, web services e computação móvel foi realizada com o intuito de familiarização com o estado da arte no que concerne às tecnologias atuais que seguem esse modelo. Em seguida, deu-se início a uma escolha de ferramentas e tecnologias que seriam utilizadas para a implementação do software. Após a escolha, a codificação da parte mobile em iOS e web em Rails foi feita, sendo implementada posteriormente em Android. Nas duas últimas etapas posteriores, testes e melhorias se deram de forma concomitante. A última etapa é reservada a melhorias e modificações demandadas por usuários, conforme o produto for sendo utilizado quando lançado no mercado, e ainda não foi inteiramente definida.

1.4 Estrutura da monografia

Esta monografia está organizada em seis capítulos, incluindo-se a Introdução aqui apresentada. O capítulo 2 aborda os conceitos gerais que serão utilizados

por todo o trabalho, como computação em nuvem, web service e tecnologias correlatas,computação móvel e os trabalhos relacionados que constituíram o embasamento teórico deste trabalho. No capítulo 3, a aplicação desenvolvida é abordada através da apresentação das ferramentas utilizadas para sua criação e descrição dos requisitos, funcionais e não-funcionais, e das camadas que compõem a arquitetura da aplicação. No capítulo 4 apresenta a tecnologia web Ruby on Rails e mostra como algumas funcionalidades foram implementadas. No capítulo 5 traça um comparativo entre as funcionalidades que foram implementadas para as plataformas iOS e Android. No Capítulo 6 apresenta a metodologia de testes que será proposta para os usuários-teste e, em seguida, realiza a apresentação e análise dos resultados obtidos. Por fim, o Capítulo 7 apresenta considerações finais acerca do trabalho realizado.

Capítulo 2

Fundamentacao

2.1 Serviços PaaS

2.1.1 Definição

Segundo o National Institute of Standarts and Technology (NIST), o termo computação em núvem é referentea "Cloud Computing é um modelo que permite de forma conveniente, o acesso à rede sob demanda para um conjunto compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e lançados com o mínimo de esforço de gestão ou a interação de um prestador de serviços.". Um dos serviços definidos é o PaaS, acrônimo do inglês "plataform as a service". Ainda de acordo com o NIST, o PaaS é "a capacidade fornecida ao consumidor de publicar aplicações usando linguagem de programação, bibliotecas, serviços suportados pelo provedor". Com o fornecimento de tal serviço, o consumidor não precisa se preocupar com o controle de certos serviços de infra-estrutura, como a rede, servidores, sistemas operacionais, armazenamento, ou seja, criam uma camada de abstração de serviços de infra-estrutura.

De acordo com `http://broadcast.rackspace.com/hosting_knowledge/whitepapers/Understanding-the-Cloud-Computing-Stack.pdf`, serviços PaaS possuem as seguintes características:

- Serviços para desenvolver, testar, publicar, hospedar e manter aplicações de forma integrada;

- Arquitetura Multi-tenant, onde vários usuários podem utilizar o mesmo ambiente de desenvolvimento;
- Construção garantindo escalabilidade, incluindo balanceamento de carga (load balancing) e replicação de dados para recuperação de falhas (failover);
- Integração com webs services e banco de dados através de padrões comuns;
- Suporte para desenvolvimento em equipes, podendo ter ferramentas de planejamento de projetos e de comunicação;
- Ferramentas para gerenciamento dos custos.

Ou seja, sistemas PaaS são úteis para desenvolvedores individuais e startups, pois fornecem facilidade de publicação sem os custos e complexidades de hardwares e softwares inerentes a uma aplicação web comum. `http://www.guardian.co.uk/technology/2008/apr/17/google.software`.

2.1.2 Heroku

O Heroku uma plataforma cloud de serviços PaaS, que roda sobre o Amazon EC2 (IaaS, infra-structure as a service), existente desde junho de 2007. Possui suporte para as seguintes linguagens: Ruby, Java, Node.JS, Scala, Clojure, Python e PHP. Internamente, roda sobre o sistema operacional Ubuntu. Inicialmente, o Heroku foi desenvolvido com suporte exclusivo para a linguagem Ruby. Em julho de 2011, Matz Matsumoto, criador do Ruby, entrou para a empresa como Arquiteto-chefe e, nesse mesmo mês, passou a dar suporte também para Node.js e Clojure. Em setembro de 2011, a rede social Facebook fez uma parceria com o Heroku a fim de facilitar a publicação de aplicativos para sua própria plataforma `https://developers.facebook.com/blog/post/558`. Em poucos passos, é possível criar uma aplicação no Facebook e no Heroku, simultaneamente. O Heroku usa uma unidade de máquina virtual chamada "Dyno" com 4 cores e até 512mb de RAM.

2.2 REST

REST, acrônimo de Representional State Transfer, foi definido em uma tese de doutorado por Roy Fielding da seguinte maneira: "A REST (Transferência do Estado Representativo) é pretendida como uma imagem do design da aplicação se comportará: uma rede de websites (um estado virtual), onde o usuário progride com

uma aplicação selecionando as ligações (transições do estado), tendo como resultado a página seguinte (que representa o estado seguinte da aplicação) que está sendo transferida ao usuário e apresentada para seu uso."

De modo geral, o REST é uma interface de comunicação onde há um provedor de serviços e um consumidor. Tal interface pode ser descrita utilizando XML, HTTP, YAML, JSON ou até mesmo texto puro, de modo a não utilizar trocas de mensagens complexas como o SOAP.

O REST possui alguns princípios, a saber:

- Modelo provedor/consumidor Stateless (sem estado): cada mensagem HTTP trocada possui todas as informações necessárias para a comunicação, ou seja, nenhuma das partes necessita gravar estado da comunicação. Em sistemas web, é comum o uso de cookies para manter o estado da sessão; já em sistemas mobile, é comum utilizarmos um token de autenticação, com a mesma finalidade.

- Operações HTTP: de modo a diminuir o tráfego de dados, são utilizados métodos HTTP para acessar os recursos de informação. As operações mais utilizadas são o GET, PUT, POST e DELETE. Em sistemas REST, é comum combinar tais métodos com operações de CRUD (create, read, update e delete), que faz persistência de dados em um determinado recurso ou entidade.

- Identificação de recursos: As URIs identificam cada uma das entidades e seus elementos, ficando a cargo da operação HTTP definir a ação a ser feita com cada um dos recursos ou elementos.

- Uso de hipermídia: As trocas de mensagem de comunicação é feita utilizando, no corpo da mensagem HTTP, uma linguagem de marcação, conforme já citado anteriormente. Porém, não há uma restrição geral quanto ao uso, podendo ser usada linguagens próprias (texto puro).

2.2.1 RESTful

Em uma arquitetura julgada como RESTful, o método desejado é informado dentro do método HTTP, contido no header do mesmo. Além disso, o escopo da informação é colocado na URL, o que, de acordo com 1, torna uma "combinação poderosa". De acordo com 1, por definição, uma aplicação deixa de ser RESTFul

caso o método HTTP não combine com o método da informação, ou seja, com a funcionalidade esperada para aquela estrutura de dados.

2.3 JSON

2.3.1 Introdução

JSON, ou JavaScript Object Notation, é um padrão aberto de texto para representar estruturas de dados, de forma inteligível para humanos. Sua origem é a linguagem javascript, e seu formato está descrtio no RFC 4627.

2.3.2 Definição

O JSON é um dos formatos mais usados na serialização e transmissão de dados estruturados pela internet,ao lado do XML e YAML, sendo muito usado em sistemas orientados a serviço / webservice. Muitas linguagens e frameworks, como o Foundation (iOS) e o Android, dão suporte para esse padrão, através de parsers para construção e consumo. De acordo com 1, "é muito mais fácil para um browser lidar com uma estrutura javascript oriunda de uma estrutura JSON do que a partir de um documento XML". Ainda de acordo com 1, cada web browser oferece uma interface JavaScript diferente para seus parsers XML, enquanto um objeto JSON, que por definição é um objeto JavaScript, será interpretado da mesma maneira em qualquer interpretador JavaScript. De acordo com 1, o JSON é uma alternativa mais leve para serialização de dados do que o XML, definido pelo XML Schema.

2.3.3 comparação com XML

JSON e XML são dois formatos de manipulação de informações que podem ser usados com o mesmo objetivo, mas possuem implementações e aplicações distintas. No artigo "Comparison of JSON and XML Data Interchange Formats: A Case Study ", é feito um estudo considerando a hipótese de que não há diferença em relação ao tempo de transmissão e os recursos utilizados entre JSON e XML.A fim de realizar o estudo, foi criado um ambiente operacional consistindo de uma aplicação cliente/servidor em Java, onde o servidor escuta uma porta e o cliente conecta a ela. No referido teste, foram utilizadas as seguintes métricas: número de objetos enviados, tempo total para enviar o número de objetos, tempo médio de transmissão, uso da CPU pelo usuário, uso da CPU pelo sistema e o uso de memória.

De acordo com a conclusão do artigo, codificação JSON é, em geral, mais rápida e consome menos recursos que a codificação XML, o que nega a hipótese de igualdade de escolha entre as duas tecnologias. Ou seja, em um ambiente onde é necessária velocidade e os recursos são limitados, como sistemas móveis, é preferível utilizar JSON.

2.3.4 Estrutura

De acordo com W3resource`http://www.w3resource.com/JSON` structures.php), o JSON suporta duas grandes estruturas de informação: coleção de pares chave/valor e listas ordenadas de valores. Ambas estruturas são também suportadas pela maioria das linguagens de programação modernas, o que reforça a ideia de ser uma boa escolha de linguagem para transmissão de informações. O JSON possui alguns tipos de dados, a saber:

- Objetos: um objeto começa e termina com '' e '', contendo um número de pares chave/valor. A separação entre uma chave e um valor é feita com o caractere ':', e a separação entre pares é feita com ','. O valor de um par pode ser qualquer estrutura JSON.
- Arrays: Um array começa e termina com '[' e ']'. Entre eles, são adicionados certo numero de valores, separados por ','.
- Valores: os valores podem ser string, numero, objeto, array, valor booleano ou null.

Exemplo de código JSON:

```
{
    "firstName": "Bidhan",
"lastName": "Chatterjee",
    "age": 40,
    "address": {
        "streetAddress": "144 J B Hazra Road",
        "city": "Burdwan",
        "state": "Paschimbanga",
        "postalCode": "713102"
    },
```

```
    "phoneNumber": [
        {
            "type": "personal",
            "number": "09832209761"
        },
        {
            "type": "fax",
            "number": "91-342-2567692"
        }
    ]
}
```

2.4 Ruby on Rails

2.4.1 Ruby

Ruby é uma linguagem orientada a objetos, com tipagem forte e dinâmica, criada por Yukihiro Matsumoto (Matz) em 1995. Segundo a apostila da Caelum de ruby on rails, uma de suas principais características é sua expressividade, ou seja, a facilidade de ser lida e entendida, o que facilitaria o desenvolvimento de sistemas escritos por ela. O livro Learning rails 3, da editora Oreilley, cita algumas das características mais importantes do Ruby, a saber:

- É uma linguagem interpretada, ou seja, um interpretador lê o código e decide como executar em tempo de execução. Por consequência, um sistema em Ruby pode se tornar um pouco mais lento, porém, é notável o ganho em flexibilidade.
- Possui uma sintaxe de linguagem flexível, fazendo com que a curva de aprendizado seja menor em relação a outras linguagens. Um exemplo clássico é a não-necessidade (opcional) de escrever parênteses ao redor dos parâmetros de um método. Entretanto, podem surgir erros misteriosos por não colocar parênteses em algumas situações ambíguas.
- Possui uma tipagem dinâmica, ou seja, não é necessário especificar o tipo de informação que será guardado em cada variável. Isso torna a abordagem bem mais flexível, tornando as operações dependentes do próprio contexto.

Entretanto, problemas podem ocorrer justamente por isso: comportamentos inesperados.

- Suporte a blocos e closures,

Atualmente, Ruby encontra-se entre as linguagens de programação mais populares, ocupando a posição 11º no índice Tiobe. Grande parte desse sucesso deve-se ao framework Rails, implementado como solução web utilizando Ruby.

2.4.2 Rails

O Ruby on Rails, também chamado Rails ou RoR, é um framework de desenvolvimento web de código aberto que tem como premissa aumentar a velocidade e a facilidade no desenvolvimento de aplicações web orientados a banco de dados. Foi lançado oficialmente em Julho de 2004 pelo seu criador David H. Hansson, estando atualmente na versão 3.2.11, com a 4ª versão em desenvolvimento. `http://blog.wyeworks.com/2012/10/29/rails-4-in-30-minutes/` O Rails é um framework full-stack, ou em português, pilha completa. Isso significa que, com ele, é possível desenvolver a aplicação por completo, desde o desenvolvimento dos layouts das paginas, a manutenção do banco de dados. Além disso, o Rails enfatiza o uso de alguns padrões de engenharia de software, a saber:

- Active record: padrão de projeto para armazenamento de dados em banco de dados relacionais. A interface de um certo objeto deve incluir funções de CRUD, como inserir, atualizar, apagar e algumas funções de consulta. Cada tabela de um banco de dados é embrulhada (wrapped) em um uma classe, sendo cada instancia dessa classe um registro (tupla) único na tabela. Esse conceito está definido em FOWLER, Martin. Patterns of enterprise application architecture.

- Convenção sobre configuração: modelo de desenvolvimento de software que busca diminuir o número de decisões que os desenvolvedores precisam tomar, ou seja, o desenvolvedor não precisa definir aspectos convencionais da aplicação.

- DRY (dont repeat yourself): O objetivo principal é reduzir a repetição de informação de qualquer tipo. Esse conceito incentiva o bom uso da reutilização de código, que é também uma das principais vantagens da orientação a objetos.

- MVC (model-view-controller): esse padrão de arquitetura de software separa a aplicação em três camadas: uma contendo a lógica da aplicação e regra de negócios, chamada model; uma contendo a entrada e saída de dados com o usuário, chamada view; uma interligando ambas, de maneira a manipular dados da view para o model entender e vice-versa, chamada controller. O principal objetivo dessa arquitetura é a reusabilidade de código e a separação de conceitos.

`http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices`
De acordo com a apostila da Caelum, a estrutura a qual o Rails é feito permite que as funcionalidades de um sistema possam ser implementadas de maneira incremental, por conta dos padrões e conceitos supracitados. Por conseqüência, isso tornaria o Rails uma boa escolha para projetos e empresas que adotam metodologias ágeis no desenvolvimento da aplicação.

O Ruby é uma linguagem interpretada. Antes de se tornar popular, existia apenas um interpretador disponível, escrito em C pelo próprio criador da linguagem. Hoje em dia, o interpretador mais conhecido é o 1.9 ou YARV (Yet Another Ruby VM), para a versão mais atualizada e estável (Ruby 1.9.3.). Existem outros interpretadores Ruby famosos, como:

- JRuby implementação alternativa que permite usar a JVM do Java para interpetar código Ruby. Uma de suas principais vantagens é a interoperabilidade com código Java existente, além de aproveitar as vantagens já maduras do java: garbage collector, threas nativas, etc.
- IronRuby Implementação .Net da linguagem, mantido pela própria Microsoft
- Rubinius Traz idéias de máquinas virtuais do SmallTalk e é implementada em C/C++.

2.5 Smartphones

O mercado de smartphones está crescendo cada vez mais. Estima-se que no ínicio de 2012 o número de celulares inteligentes tenha atingido a marca de 1 bilhão de unidades vendidas e, segundo projeções, esse número deve dobrar em 2015. `http://finance.yahoo.com/news/number-smartphones-around-world-top-122000896.html` Embora o número

de smartphones seja expressivo, ele ainda é pequeno se comparado ao número de pessoas que possuem um aparelho celular: 3 bilhões. Isso significa que ainda há muito espaço para crescimento, em particular em mercados emergentes como a China, Índia e África. Usualmente, um smartphone possui alguns recursos de ponta, como câmera, bom reprodutor de mídia, bom processamento gráfico para jogos, bluetooth, GPS, acesso a internet via wi-fi e 3G/4G, NFC, um bom sistema operacional, entre outros. Atualmente, os sistemas operacionais mais populares para smartphones são: Android, iOS (Apple), Blackberry OS (RIM), Bada (Samsung), Symbian e Windows Phone 7 (Microsoft). A distribuição do mercado, no Q3 de 2012, pode ser vista no seguinte gráfico:

Fonte: `http://www.neowin.net/news/gartner-microsoft-increases-mobile-os-market-sh`
Notadamente, o Android e o iOS são as plataformas mais populares. O grande diferencial entre o marketshare desses dois sistemas são o segmento de mercado: enquanto o Android atua em todos os segmentos, desde celulares Low-end aos celulares de ponta, o iOS atua somente com celulares de ponta, chamados High-End. Outro fato a considerar é que, nesse gráfico, não são considerados outros dispositivos, como tocadores de música e tablets.

2.6 iOS

2.6.1 Visão Geral

O iOS é um sistema operacional para dispositivos móveis, lançado pela Apple em 2007. Inicialmente, foi desenvolvido para o iPhone, sendo posteriormente aproveitado nos dispositivos iPod Touch, iPad e Apple TV. Ele é um sistema operacional licenciado para rodar apenas em hardware produzido pela Apple, otimizado para a arquitetura de processadores ARM. Sua entrada de dados é feita de forma direta, através de multi- toques. Esses toques podem ser desde encostar o dedo, similar a um clique do mouse, até balançar o aparelho, de modo a utilizar seu acelerômetro. Todos os controles de entrada de dados são controlados pela GUI Cocoa Touch. O livro "Use a cabeça Desenvolvimento para iPhone" cita que o iPhone revolucionou a maneira de ver um celular: ele é, atualmente, uma plataforma de jogos, um organizador pessoal, um navegador (browser) completo e, claro, um celular. Muito de seu sucesso deve-se ao sucesso da loja virtual "App Store", de mídias e aplicativos, que abriu oportunidade para desenvolvedores independentes

competirem em escala mundial com grandes empresas de software.

2.6.2 Linguagem: Objective-c

A programação nativa em iOS utiliza uma linguagem de programação chamada Objective-C, com o framework Foundation. O Objective-C é uma linguagem de programação reflexiva e orientada a objeto, com origens no SmallTalk e no C. Foi criada no início da década de 80 por Brad Cox e Tom Love, mas somente se tornou popular quando foi licenciada pela NeXT, de Steve Jobs, em 1988. Atualmente é a principal linguagem utilizada para desenvolvimento para Mac OS X. Como o Objective-C foi construido sobre C, qualquer código C pode ser compilado com um compilador Objective-C. Por definição, é uma camada sobre o C que aceita orientação a objetos, através de mensagens. (adicionar referencia) Em linguagens com "message parsing", métodos não são chamados de objetos, mas sim mensagens são enviadas ao objeto. Essa diferença implica em como o código referenciado pelo método ou nome da mensagem é executado. Em nosso caso, o "alvo" da mensagem é resolvido em tempo de execução, com o objeto receptor interpretando a mensagem.

2.6.3 Ciclo de vida

O ciclo de vida constitui uma sequência de eventos entre o início e a finalização da aplicação. Um aplicativo iOS começa quando o usuário toca o ícone da mesma na home do dispositivo. Feito isso, o sistema operacional inicia alguns procedimentos de renderização e chama a função principal (main.m) do aplicativo. Uma vez iniciado, o comando da execução passa a ser do UIKit, framework de controle do iOS, que carrega a interface gráfica e lê o loop de eventos. Durante o loop, o UIKit delega cada evento a seu respectivo objeto e responde aos comandos emitidos pelo aplicativo. Quando o usuário realiza uma ação que causa um evento de saída, o UIKit notifica a aplicação e inicia o processo de saída.

2.7 Android

2.7.1 Visão Geral

O Android é a resposta do Google para ocupar o segmento de mercado de sistemas operacionais para plataformas móveis. Consiste em um novo sistema baseado no sistema operacional Linux, com diversas aplicações já instaladas, além

de um ambiente de desenvolvimento forte e flexível. De acordo com "Google Android Lecheta", o Android causou um grande impacto quando foi anunciado, em especial pelas empresas que estavam por trás de seu desenvolvimento: Google, Motorola, LG, Samsung, Sony, entre muitas outras. A esse grupo de empresas, denominado Open Handset Alliance (OHA), coube à padronização de uma plataforma de código aberto e livre para celulares, com o objetivo de atender a demanda do mercado atual. Um dos pontos fortes do Android é seu sistema flexível: é fácil integrar aplicações nativas com a sua aplicação, ou até mesmo substituir algumas dessas aplicações nativas pela sua própria. Isso gera um grande apelo para empresas de telefonia, que podem usar dessa personalização para lançarem suas próprias versões de aparelhos Android personalizados. O grande foco do sistema é a interação entre aplicativos: agenda, maps, contatos são facilmente alcançáveis por qualquer aplicação. Essa funcionalidade é realizada por um recurso chamado Intent, que será abordado em capítulos posteriores. Outro ponto forte do Android é que seu sistema operacional é baseado no Linux (baseado no kernel 2.6), ou seja, ele mesmo se encarrega de gerenciar a memoria em uso e os processos. Isso faz com que seja possível rodar mais de uma aplicação (genuinamente) ao mesmo tempo, fazendo com que outros aplicativos rodem em segundo plano durante outros serviços, como ao atender um telefonema ou acessar a internet. O que é dito como vantagem também é apontado, fatalmente, como um problema: uma vez que aplicativos podem rodar em segundo plano, aplicativos maliciosos também podem ser rodados em segundo plano. Durante muito tempo, aplicativos desse tipo podiam ser encontrados para download no Android Market (atualmente Google Play), havendo hoje uma melhor seleção dos aplicativos que de fato estão sendo disponibilizados na loja.

2.7.2 Linguagem: Java

A linguagem nativa de programação para Android é o Java, utilizando o framework Android criado pela Open Handset Alliance (OHA). O Java é uma linguagem de propósito geral, concorrente e orientada a objetos. Sua primeira versão foi lançada em 1995 pela Sun Microsystems e, atualmente, encontra-se em sua sétima versão, sendo mantida pela Oracle. Atualmente, é uma das linguagens de programação mais populares do mundo, ocupando o 2º lugar no índice TIOBE. Devido a isso, há um grande número de desenvolvedores que possuem o pré-requisito básico para iniciar programação para Android: o domínio da linguagem. O grande sucesso do Java é normalmente creditado a sua capacidade de funcionar nos

mais diversos ambientes, desde micro-sistemas como cartões de crédito a grandes plataformas web. Isso é devido a implementação de sua máquina virtual, que é capaz de rodar nas mais diversas plataformas.

2.7.3 Ciclo de vida

O ciclo de vida de uma aplicação Android é controlado por uma Activity, a qual também gerencia a interface com o usuário, recebe requisições, realiza o tratamento e processa. Toda Activity possui os seguintes métodos de controle, a saber:

- onCreate(): é o primeiro método a ser executado em uma Activity. Usualmente é o método responsável por carregar os layouts XML e inicializar atributos de classe e outros serviços.
- onStart(): é chamado imediatamente chamado após o onCreate(). Diferentemente deste, é chamado toda vez que a Activity volta a ter foco após um período em background.
- onResume(): Assim como o onStart(), é chamado no início da Activity e, também, quando a mesma volta a ter foco. A diferença entre ambos é que o onStart() só é invocado quando a Activity não está mais visível, enquanto o onResume() é chamado toda vez que retorna o foco.
- onPause(): é a primeira função a ser chamada quando a Activity perde o foco.
- onStop(): é chamado quando uma Activity é substituída por outra Activity
- onDestroy(): é o último método a ser executado. Quando o onDestroy() é executado, a Activity é considerada "morta" e fica pronta para ser removida pelo Garbage Collector. onRestart(): Chamado quando a Activity sai do estado de "stop"; após sua execução, é chamado o método onStart().

2.8 Redes Sociais

será que vale mesmo a pena falar sobre isso?

2.9 Resumo do Capítulo

Neste capítulo foram abordadas as três principais tecnologias necessárias para o desenvolvimento deste projeto: computação em nuvem, web services e computação

móvel. Tais tecnologias foram aprofundadas em um nível o qual fornecessem ao leitor os pré-requisitos para a compreensão do restante do projeto, o qual será apresentado nos capítulos posteriores.

Capítulo 3

Aplicação

3.1 Visão geral

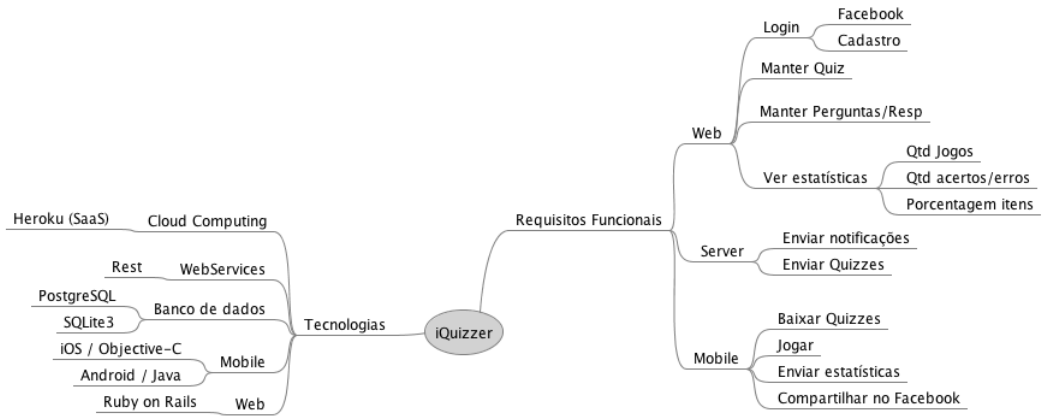

Figura 3.1: Mapa Mental

3.2 Requisitos

3.2.1 Requisitos funcionais

3.2.2 Requisitos não-funcionais

3.3 Casos de uso

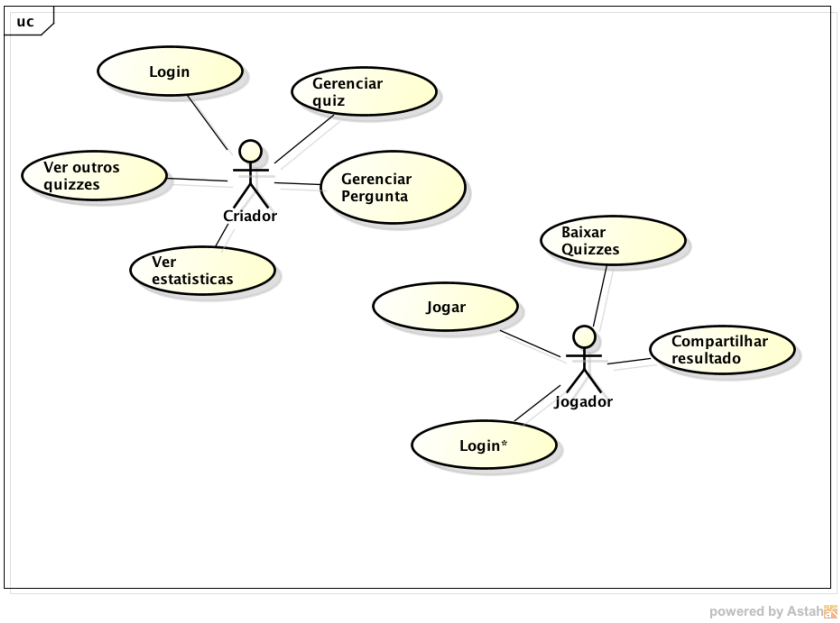

Figura 3.2: Diagrama de casos de uso para Criador e Jogador

3.4 Diagrama de classes

3.5 Diagrama de entidades

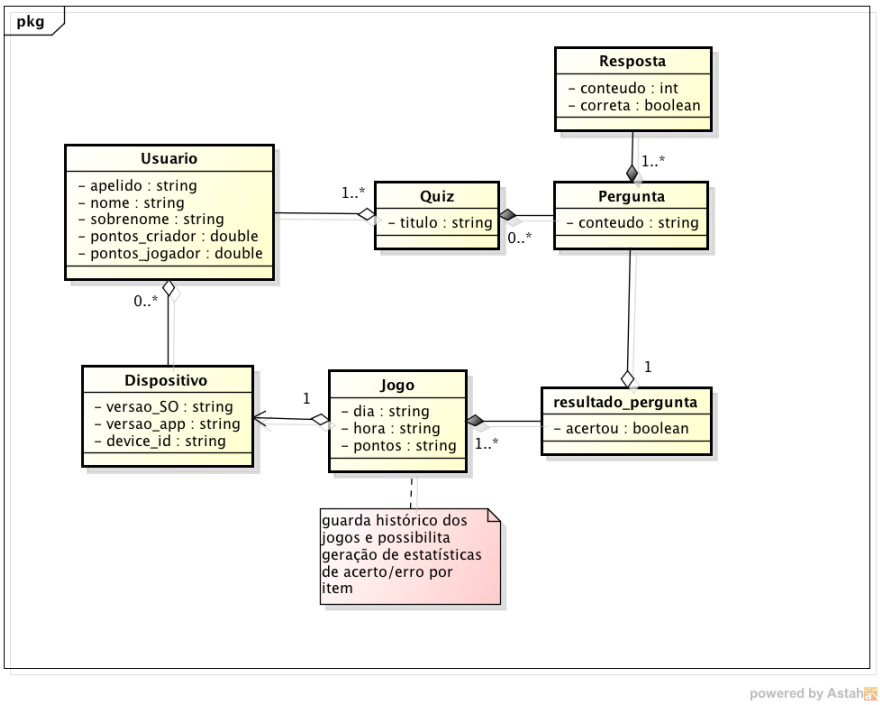

Figura 3.3: Diagrama de entidades

Capítulo 4

Implementação web

4.1 Criação de aplicação

4.2 Rotas

4.3 Autenticação

4.4 Front-End com bootstrap

4.5 Deploy

Capítulo 5

Comparativo Android x iOS

5.1 Ambientação

5.1.1 Linguagens

5.1.2 IDEs

5.2 Arquitetura

5.2.1 Arquivos gerados

5.3 Controles básicos

5.3.1 Mostrando textos

5.3.2 Inserindo textos

5.3.3 Capturando eventos de botões

5.4 Criando listas

5.4.1 Listando arrays

5.4.2 Personalizando linhas

5.4.3 Capturando eventos de seleção

5.5 Acesso a dados

5.5.1 SQLite

5.5.2 Preferências

5.6 Parser JSON

5.7 Conexão HTTP

Capítulo 6

Resultados

Capítulo 7

Conclusão

7.1 Perspectivas Futuras

Referências Bibliográficas

ADAMS, R. *High performance memory testing : design principles, fault modeling, and self-test*. Boston: Kluwer Academic, 2003. ISBN 1402072554.

ARCHIVES, I. *Magnetic Disk*. 1970. Disponível via Internet, <`http://www-03.ibm.com/ibm/history/exhibits/storage/storage_PH4-15.html`>. Acessado em 15 de Outubro de 2011.

BEZ, R. *et al.* Introduction to flash memory. *Proceedings of the IEEE*, v. 91, n. 4, p. 489 – 502, april 2003. ISSN 0018-9219.

CARRIER, B. *File system forensic analysis*. Boston, Mass. London: Addison-Wesley, 2005. ISBN 0321268172.

COMMITTEE, I. C. for I. T. S. I. T. T. *AT Attachment 8 - ATA/ATAPI Architecture Model(ATA8-AAM)*. Projeto t13/1700-d. [S.l.], Maio 2006.

COMMITTEE, I. C. for I. T. S. I. T. T. *AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS))*. Projeto t13/1699-d. [S.l.], Maio 2007.

COMMITTEE, I. C. for I. T. S. I. T. T. *SCSI / ATA Translation - 2 (SAT-2)*. Projeto t10/1826-d. [S.l.], Junho 2008.

COMMITTEE, I. C. for I. T. S. I. T. T. *SCSI Architecture Model - 5 (SAM-5)*. Projeto t10/2104-d. [S.l.], Setembro 2008.

COMMITTEE, I. C. for I. T. S. I. T. T. *T10 Working Drafts*. 2011. Acessado em 15 de Agosto de 20111. Disponível em: <http://www.t10.org/drafts.htm>.

COMPONENTS, I. T. A. E. *Solid State Drives, Separating myths from facts*. Junho 2009. Disponível via Internet, <`http://www.toshiba.com/taec/components/Generic/SSD_Myths.pdf`>. Acessado em 04 de Novembro de 2011.

CORPORATION, D. A. *Hard Disk Sector Structures*. Dezembro 2001. Disponível via Internet, <`http://www.dewassoc.com/kbase/hard_drives/hard_disk_sector_structures.htm`>. Acessado em 02 de Novembro de 2011.

EIWFELDT, H. *The Linux SCSI programming HOWTO*. Maio 1996. Disponível via Internet, <`http://tldp.org/HOWTO/archived/SCSI-Programming-HOWTO/`>. Acessado em 20 de Maio de 2011.

HARKER, J. M. *et al.* A quarter century of disk file innovation. *IBM Journal of Research and Development*, v. 25, n. 5, p. 677 –690, sep. 1981. ISSN 0018-8646.

HUGHES, G. *et al.* Improved disk-drive failure warnings. *Reliability, IEEE Transactions on*, v. 51, n. 3, p. 350 – 357, sep 2002. ISSN 0018-9529.

INTEL. *What are the advantages of TRIM and how can I use it with my SSD?* Setembro 2010. Disponível via Internet, <`http://www.intel.com/support/ssdc/hpssd/sb/CS-031846.htm?wapkw=(TRIM)`>. Acessado em 04 de Novembro de 2011.

ISERMANN, R. *Fault-diagnosis applications : model-based condition monitoring : actuators, drives, machinery, plants, sensors, and fault-tolerant systems*. Heidelberg New York: Springer, 2011. ISBN 9783642127663.

KARI, H. *Latent sector faults and reliability of disk arrays*. Helsinki University of Technology, 1997. ISBN 9789512235483. Disponível em: <http://books.google.com/books?id=pspEAAAACAAJ>.

KINNEY, M. H. A. *Generic SCSI Command Generator*. Dissertação — Worcester Polytechnic Institute, Março 2004. Disponível em: <http://web.cs.wpi.edu/~claypool/mqp/gd-scsi/final.pdf>.

KUNETT, V. *et al.* An in-system reprogrammable 256k cmos flash memory. In: *Solid-State Circuits Conference, 1988. Digest of Technical Papers. ISSCC. 1988 IEEE International*. [S.l.: s.n.], 1988. p. 132.

LEE, P.; ANDERSON, T. *Fault tolerance, principles and practice*. Wien New York: Springer-Verlag, 1990. ISBN 0387820779.

LOUGHE, P. *An Overview of the SquashFS filesystem*. 2008. Disponível via Internet, <`http://tree.celinuxforum.org/CelfPubWiki/ELCEurope2008Presentations?action=AttachFile&do=get&target=squashfs-elce.pdf`>. Acessado em Novembro de 2011.

MAMUN, A. *Hard disk drive : mechatronics and control*. Boca Raton, FL: CRC Press, 2007. ISBN 9780849372537.

MASON, H. Scsi, the industry workhorse, is still working hard. *Computer*, v. 33, n. 12, p. 152 – 153, dec 2000. ISSN 0018-9162.

MASUOKA, F. *et al.* A new flash e2prom cell using triple polysilicon technology. In: *Electron Devices Meeting, 1984 International*. [S.l.: s.n.], 1984. v. 30, p. 464 – 467.

MCLEAN, P. *Proposal for a Selective Self-test*. Outubro 2001. Disponível via Internet, <`http://www.t10.org/t13/technical/e01139r0.pdf`>. Acessado em 15 de setembro de 2011.

MORIMOTO, C. *Hardware - O Guia Definitivo*. SULINA, 2007. ISBN 9788599593103. Disponível em: <http://books.google.com/books?id=P0ZDbwAACAAJ>.

MORIMOTO, C. *Hardware, V.2 - O Guia Definitivo*. SULINA, 2010. ISBN 9788599593165. Disponível em: <http://books.google.com/books?id=vPpBbwAACAAJ>.

MORIMOTO, C. E. *Ciclos de Gravação e a questão da logenvidade*. Outubro 2010. Disponível via Internet, <`http://www.hardware.com.br/tutoriais/entendendo-ssd/ciclos-gravacao-longevidade.html`>. Acessado em 20 de Outubro de 2011.

PARRISH, K. *Seagate Launching "Industry's First" 4TB HDD*. Setembro 2011. Disponível via Internet, <`http://www.tomshardware.com/news/GoFlex-Desk-4TB-3.5-inch-industrial-design-USB-3.0,13371.html`>. Acessado em 02 de Novembro de 2011.

PC-DOCTOR. *Case Study, PC-Doctor Hard Drive Testing*. 2011. Disponível via Internet, <http://www.pc-doctor.com/files/english/case_study_hdd_testing.pdf>. Acessado em Julho de 2011.

POTTS, P. R. *SixHardDriveFormFactors.jpg*. Março 2008. Disponível via Internet, <http://en.wikipedia.org/wiki/File:SixHardDriveFormFactors.jpg>. Six hard disk drives with cases opened showing platters and heads; 8, 5.25, 3.5, 2.5, 1.8 and 1 inch disk diameters are represented.

RENT, T. M. *Origins of Solid State Drives*. Março 2010. Disponível via Internet, <http://www.storagereview.com/origin-solid-state-drives>. Acessado em 12 de Outubro de 2011.

RENT, T. M. *SSD Architecture*. Abril 2010. Disponível via Internet, <http://www.storagereview.com/ssd_architecture>. Acessado em 20 de Agosto de 2011.

RENT, T. M. *SSD Controller*. Abril 2010. Disponível via Internet, <http://www.storagereview.com/ssd_controller>. Acessado em 12 de Outubro de 2011.

RICHARDS, M. *Card Capacitor Read Only Store (CCROS) for microprogram*. 1965. Disponível via Internet, <http://www.computerhistory.org/revolution/memory-storage/8/264/1103>. Acessado em 22 de Outubro de 2011.

SEAGATE. *The Case for Solid-State Hybrid Drives*. 2011. White Paper. Disponível via Internet, <http://www.seagate.com/docs/pdf/corporate/seagate_view_solid_state_hybrid_drive_gen.pdf>. Acessado em 14 de Setembro de 2011.

SHIMPI, A. L. *OCZ Z-Drive R4 CM88 (1.6TB PCIe SSD) Review*. Setembro 2011. Disponível via Internet, <http://www.anandtech.com/show/4879/ocz-zdrive-r4-cm88-16tb-pcie-ssd-review>. Acessado em 02 de novembro de 2011.

STEVENS, C. E. *ATA Command Pass-Through*. 04-262r8a. ed. [S.l.], Março 2005.

STEVENS, L. D. The evolution of magnetic storage. *IBM Journal of Research and Development*, v. 25, n. 5, p. 663 –676, sep. 1981. ISSN 0018-8646.

STROM, B. *et al.* Hard disk drive reliability modeling and failure prediction. *Magnetics, IEEE Transactions on*, v. 43, n. 9, p. 3676 –3684, sept. 2007. ISSN 0018-9464.

SYSADM, H. *Solid State Drives, some theory and a selection of videos*. Julho 2011. Disponível via Internet, <`http://www.happysysadm.com/search?q=ssd`>. Acessado em 20 de Agosto de 2011.

TANENBAUM, A. *Modern operating systems*. Upper Saddle River, N.J: Prentice Hall, 2001. ISBN 0130313580.

TAO, M. Y. *Tour the Linux generic SCSI driver*. Fevereiro 2009. Disponível via Internet, <`http://www.ibm.com/developerworks/linux/library/l-scsi-api/index.html`>. Acessado em 8 de maio de 2011.

TOOLBOX, L. S. *SCSI TOOLBOX, LLC - Using SAT to Access SATA drives*. Agosto 2008. Disponível via Internet, <`http://www.scsitoolbox.com/pdfs/IssuingATACommands.pdf`>. Acessado em 28 de Julho de 2011.

TROJANOWSKI, B. *ATA messages via SCSI Layer*. Janeiro 2010. Disponível via Internet, <`http://www.jukie.net/bart/blog/ata-via-scsi`>. Acessado em 02 de Novembro de 2011.

VASCONCELOS, L. *Hardware Total*. [S.l.]: MAKRON BOOKS, 2002.

WESZ, R. *Falha, erro ou defeito?* Junho 2007. Disponível via Internet, <`http://testsnews.wordpress.com/2007/06/24/falha-erro-ou-defeito/`>. Acessado em 20 de Outubro de 2011.

WIKI, U. *BootToRam*. Julho 2011. Disponível via Internet, <`https://wiki.ubuntu.com/BootToRAM`>. Acessado em Julho de 2011.

ZHAO, X. *et al.* A novel method to restrain the radial error propagation in self-servowriting in hard disk. In: *Automation and Logistics, 2007 IEEE International Conference on*. [S.l.: s.n.], 2007. p. 2736 –2739.

ZIELSKI, S.; SOSNOWSKI, J. The scsi interface conformance tests generation. In: *Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07. 2nd International Conference on*. [S.l.: s.n.], 2007. p. 360 –367.