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

**Tiago Augusto da Silva Bencardino**

# iQuizzer: Estudo de caso para integração entre aplicações Web e dispositivos móveis

Fortaleza – Ceará
Fevereiro 2013

**Autor:**
Tiago Augusto da Silva Bencardino

**Orientador:**
Prof. Dr. José Marques Soares

# iQuizzer: Estudo de caso para integração entre aplicações Web e dispositivos móveis

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

**iQuizzer: Estudo de caso para integração entre aplicações Web e dispositivos móveis**

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, 6 de fevereiro 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

**CSS** *Cascading Style Sheets*

**DRY** *don't repeat yourself*

**GUI** *Graphic User Interface*

**HTML** *HyperText Markup Language*

**HTTP** *Hypertext Transfer Protocol*

**JSON** *JavaScript Object Notation*

**JVM** *Java Virtual Machine*

**IDE** *Integrated Development Environment*

**MVC** *Model View Controller*

**NFC** *Near Field Communication*

**NIST** *National Institute of Standarts and Technology*

**OHA** *Open Handset Alliance*

**PaaS** *Platform as a Service*

**REST** *Representational State Transfer*

**SGBD** *Sistema de Gerenciamento de Banco de Dados*

**SOAP** *Simple Object Access Protocol*

**SSH** *Secure Shell*

**TDD** *Test Driven Development*

**URL** *Uniform Resource Locator*

**XML** *Extensible Markup Language*

**YAML** *YAML Ain't Markup Language*

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 *Simple Object Access Protocol* (SOAP) e *Representational State Transfer* (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 (**??**). 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, de forma competitiva ou apenas responder, a a depender do modo de jogo inerente ao quiz em questão.

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 aplicativos móveis;

**v.** Implementação mobile da aplicação;

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

**vii.** Documentação de referência para projetos de arquitetura similar.

## 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*.

Por fim, o capítulo 7 apresenta considerações finais acerca do trabalho realizado.

Capítulo 2

# Fundamentação Teórica

## 2.1 Serviços PaaS

### 2.1.1 Definição

Segundo o *National Institute of Standarts and Technology* (NIST), o termo computação em núvem tem a seguinte definição: "Cloud Computing (pt: computação em nuvem) 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 *Platform as a Service* (PaaS).

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 (**??**), 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 web 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[1], pois fornecem facilidade de publicação sem os custos e complexidades de hardwares e softwares inerentes a uma aplicação web comum (**??**).

### 2.1.2 Heroku

O *Heroku* é uma plataforma cloud de serviços PaaS montado sobre o *Amazon EC2*, existente desde junho de 2007. Possui suporte para as seguintes linguagens: Ruby, Java, Node.JS, Scala, Clojure, Python e PHP. Internamente, funciona 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 (**??**). 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

O REST foi definido em uma tese de doutorado por Roy Fielding da seguinte maneira: "O REST é pretendido como uma imagem do design da aplicação se

[1]Termo utilizado para designar modelo de négocios repetível e escalável, em um ambiente de extrema incerteza. Normalmente, projetos ou empresas startups estão associadas as áreas de tecnologia.

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 *Extensible Markup Language* (XML), *Hypertext Transfer Protocol* (HTTP), *YAML Ain't Markup Language* (YAML), *JavaScript Object Notation* (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 URLs 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 *Uniform Resource Locator* (URL), o que torna uma "combinação poderosa". 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 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 (**??**), "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 a fonte, 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. 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 (**??**), 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 (**??**), 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 JSONpossui 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 (**??**), 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 (**??**) lista 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[2]. 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 (**??**).

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 (**??**).

- 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

[2]O índice TIOBE mede, mensalmente, a popularidade de uma determinada linguagem de programação, baseado no numero de engenheiros qualificados, cursos, vendedores e buscas nos principais motores de busca. Pode ser encontrado em `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

da aplicação. Em Rails, é fácil perceber esse padrão na escolha dos nomes das tabelas: se um modelo chama-se "Usuario", a tabela correspondente se chamará "Usuarios" e, se existir uma relação entre usuários e contas (m:m), a nova tabela será denominada, automaticamente, $usuarios_contas$.

- *don't repeat yourself* (DRY): 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.

- *Model View Controller* (MVC): 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 (**??**).

De acordo com (**??**), 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 *Java Virtual Machine* (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 (**??**).

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, *Near Field Communication* (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:

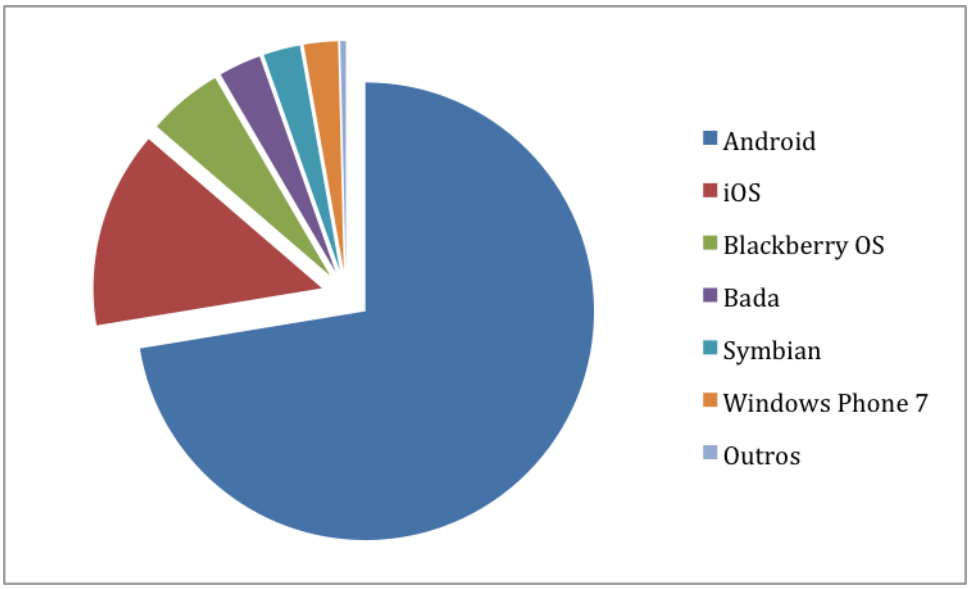


Figura 2.1: MarketShare Q3 2012

Fonte: (**??**)

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 *Graphic User Interface* (GUI) Cocoa Touch.

O livro (**??**) 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.

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 (**??**), 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.

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

Com o intuito de demonstrar a arquitetura em estudo definida nos capítulos **??** e **??**, foi idealizado um aplicativo com módulos Web e Mobile, de forma a demonstrar todas as tecnologias citadas.

Um quiz é uma forma de jogo ou esporte mental, onde os jogadores têm por objetivo responder determinadas questões corretamente. Usualmente, quizzes são usados em educação e entretenimento para medir grau de conhecimento e habilidades, como pensamento rápido e associação mental. Em modalidades mais competitivas, os quizzes podem ter pontuações e, para competições em grupo, pode ter um vencedor (o participante com maior pontuação, por exemplo).

O sistema é formado por dois grandes módulos:

- Sistema web: Nessa parte da aplicação, usuários podem gerenciar suas contas (metadados) e criar quizzes. No momento da criação, tal usuário é denominado "criador", e recebe uma pontuação por participação nesse processo. Nesse sistema, também é possível consultar os resultados dos jogos (ou partida) realizada sobre cada um dos quizzes.

- Sistema mobile: Nessa parte da aplicação, que está sobre as plataformas Android e iOS de forma nativa, o usuário (agora denominado "jogador"), pode baixar os quizzes de seu interesse para seu dispositivo e jogá-lo, enviando seu resultado no final para ser consultado posteriormente por um criador.

Tal sistema pode ser usado com duas finalidades básicas: entreter e realizar pesquisas. Na primeira forma, perguntas de conhecimento são criadas e jogadas casualmente, com o intuito de acertar uma pergunta. Na segunda forma, pode ser usada como uma enquete, onde o objetivo não é acertar uma determinada resposta, mas sim informar qual das opções corresponde a uma verdade do usuário.

## 3.2 Requisitos

### 3.2.1 Requisitos funcionais

Os requisitos funcionais definem ações que um sistema ou componente devem ser capazes de executar, sem levar em consideração restrições físicas. Para a aplicação iQuizzer, temos os seguintes requisitos:

- Gerenciamento de Quiz:a aplicação deve fornecer meios para que o dono do quiz possa inserir, modificar ou apagar dados relativos ao quiz, as perguntas pertencentes a ele e as respostas pertencentes a cada pergunta.
- Sistema de pontos: a aplicação deve adicionar pontos a cada quiz ou pergunta inseridos por um criador de quiz. Além disso, deve adicionar pontos relativos a cada jogo, desde que o modo de jogo do quiz executado permita acumular pontos.
- Gerenciamento de resultados: a aplicação deve permitir ao criador visualizar as respostas marcadas (acertos e erros, a depender do modo de jogo), de cada jogador, ao seu quiz.
- Gerenciamento de conta: a aplicação deve fornecer meios para que o usuário possa criar e apagar sua conta. Além disso, deve permitir ao usuário modificar seus dados cadastrais.

### 3.2.2 Requisitos não-funcionais

Os requisitos não-funcionais especificam alguns fatores relacionados ao uso da aplicação, como performance, usabilidade, confiabilidade, entre outros. Esses requisitos podem constituir restrições aos requisitos funcionais, pois são características mínimas de um software de qualidade, ficando a cargo do desenvolvedor optar ou não por atender esses requisitos. Para a aplicação iQuizzer, temos os seguintes requisitos:

- Interatividade:a aplicação deve exigir interação do usuário, de maneira rápida e intuitiva. Atividades básicas não podem demandar muitas interações.
- Confiabilidade:a aplicação não pode apresentar erros durante a execução. O sistema móvel deve prever situações de perda de conexão e situaçõesde estouro de memória.
- Integridade e privacidade dos dados: Toda a informação trocada entre o sistema móvel e o sistema web deve ser mantidas de maneira integra. Além disso, os dados do sistema web só podem ser visualizados caso haja autorização para isso, ou seja, devem existir autenticação e tokens de autenticação.

## 3.3 Casos de uso

O diagrama de casos de uso fornece um modo de descrever a visão externa do sistema e suas interações com o mundo exterior, representando uma visão de alto nível da funcionalidade do sistema mediante uma requisição do usuário.

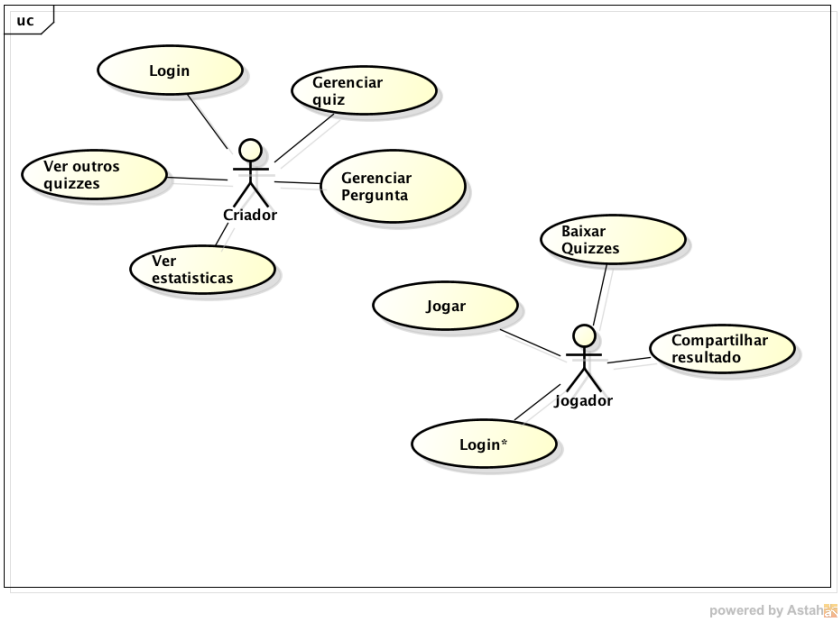


Figura 3.1: Diagrama de casos de uso para Criador e Jogador

## 3.4 Diagrama de classes

Diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para objetos. Com ela, é possível ter uma visão rápida das

classes do sistema, de modo a ajudar na implementação do modelo (M do MVC) da aplicação. Além disso, é a base para construção dos diagramas de comunicação, sequencia e estados.

Nesse diagrama, representamos nossas tabelas básicas (core) para o funcionamento da aplicação iQuizzer. Existem três entidades principais, a saber:

- Usuário: é a classe que representa o jogador e o criador do quiz. Além das informações básicas, como nome e email, estão gruardadas informações para login (username, senha) e a pontuação (de criador e jogador).
- Quiz: é a classe que representa um quiz. Hierarquicamente, um quiz possui um número ilimitado de perguntas únicas, e uma perguntapossui um número ilimitado de respostas únicas. Cada quiz possui um modo de jogo, que indica se é um quiz de perguntas arbitrarias, em sequencia, com tempo, etc, e um número máximo de perguntas a ser jogado por partida.
- Jogo: essa classe representa cada uma das partidas (jogos) que foram realizadas por cada jogador no sistema móvel. Cada jogo possui uma quantidade de resultados, correspondente ao número máximo de perguntas do quiz jogado.

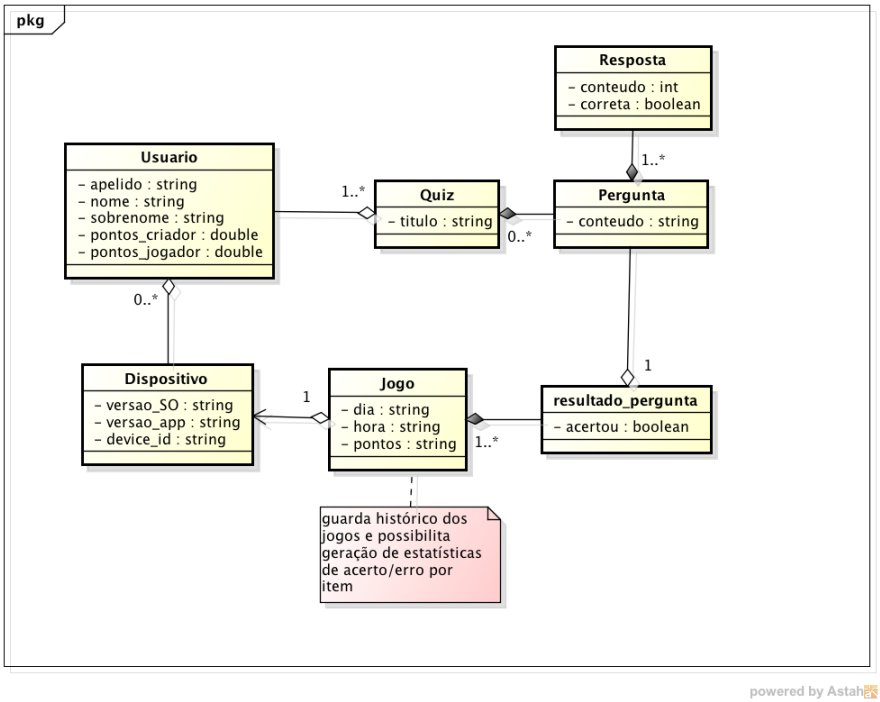


Figura 3.2: Diagrama de entidades

Capítulo 4

# Implementação web

Conforme citado no capítulo dois, utilizaremos o framework web chamado Ruby on Rails, implementado sobre a linguagem Rails.

## 4.1 Ambientação

Para a configuração do ambiente Ruby on Rails, foram utilizadas as soluções de (**??**), que implementou facilitadores para instalação em ambientes Windows e OS X. Porém, os aplicativos necessários podem ser instalados separadamente. Os necessários para a nossa aplicação foram:

- Ruby 1.9.3 -Interpretador da linguagem Ruby
- Rails 3.2.11 - Pacote contendo todo o framework Rails utilizado
- Git 1.7.10 - Controlador de versão Git. Utilizamos para o controle da versão junto ao GitHub e para fazer o deploy no Heroku
- RVM1.16.17 - Gerenciador de versões Ruby
- Bundler 1.2.1 - Controle de dependências (Gems[1])

Uma das decisões mais importantes no processo de manufatura de uma aplicação é a escolha da ferramenta ideal de desenvolvimento, que forneça agilidade e os recursos necessários para o desenvolvedor.

[1]Gems sã pacotes que contém todoa informação de arquivos a serem insalados.

Por ser uma aplicação em Ruby, onde os arquivos de código e de configuração são baseados em texto puro, o Ruby on Rails não exige nenhuma ferramenta avançada de criação. Em outras palavras, utilizar um editor de texto simples ou uma *Integrated Development Environment* (IDE) é uma decisão pessoal.

As IDE's mais comuns para se trabalhar com Rails são:

- RubyMine - Intellij IDEA
- Aptana Studio 3 - antes conhecida como RadRails, plugin para Eclipse
- Ruby in Steel- Visual Studio
- NetBeans - (até a versão 6.9)

De acordo com (**??**), a maioria dos desenvolvedores Ruby on Rails não utiliza nenhuma IDE, apenas um bom editor de texto e um terminal (ou prompt) de comando aberto. Algumas ferramentas de texto boas para esse propósito, junto aos sistemas operacionais que suportam, são as seguintes:

- TextMate - Mac OS X
- Sublime Text - Mac OS X, Linux, Windows
- Gedit - Mac OS X, Linux
- Notepad++ - Windows
- VI/Vim - Mac OS X, Linux, Windows

Nesse projeto foi utilizado o TextMate, no ambiente Mac OS X.

## 4.2 Criação de aplicação

Uma vez que o Rails tenha sido instalado corretamente e, automaticamente, definido como variável de ambiente no terminal, pode-se executar o comando "rails" para criação de aplicação e módulos. Pode-se, também, verificar a versão do rails através do comando: rails – version

A versão utilizada nesse projeto é a 3.2.11, a última disponibilizada. A fim de criar a aplicação, utilizando o banco de dados Postgre, foi executado o seguinte comando: rails new iQuizzer -d postgresql

Ao executar a ultima linha, o rails criou um diretório com alguns arquivos, que serão vistos na próxima seção.

## 4.3 Estrutura

Um projeto rails tem uma estrutura básica composta das seguintes pastas:

- App: contém os arquivos específicos da aplicação. Conforme citado no capítulo **??**, o rails utiliza o padrão MVC, e dentro desse diretório a divisão é feita através dos sub-diretórios model, view e controller;
- Config: configurações diversas da aplicação;
- db: contém as migrações (alterações no banco de dados durante o processo de desenvolvimento), além de alguns outros arquivos relacionados ao banco de dados
- doc: documentação do sistema
- lib: bibliotecas
- log: algumas informações de log
- public: nessa pasta estão contidos todos os arquivos públicos que serão servidos pela WEB
- test: utilizado para testar a aplicação, normalmente quando se usa *Test Driven Development* (TDD)[2]. Em nosso projeto, não será utilizado.
- Tmp: arquivos temporários de sessão e cache
- Vendor: projetos dependentes (terceiros)

## 4.4 Banco de dados

### 4.4.1 PostgreSQL

Banco de dados são coleções de informações que se relacionam de forma que crie um sentido. Seu objetivo principal é representar abstratamente uma parte do mundo real, conhecida como Universo de Discurso (**??**).

[2]Desenvolvimento orientado a testes é uma técnica de desenvolvimento de software onde um caso de teste é escrito para cada funcionalidade a ser implementada. A implementação deve validar o teste e o código refatorado para padrões aceitaveis.

Na aplicação, foi utilizado um banco de dados relacional chamado PostgreSQL. Lançado em 1995, está atualmente na versão 9.1.4, sobre a licença BSD[3].

A opção pela escolha do PostgreSQL deve-se ao bom suporte do Heroku a este *Sistema de Gerenciamento de Banco de Dados* (SGBD). Entretanto, outros motivos poderiam ser colocados em prol dessa decisão, a saber:

- Independência de plataforma - roda nos principais sistemas operacionais
- Leve, podendo ser rodado em desktops convencionais
- Instalação simples
- Gratuito

### 4.4.2 Modelos

Conforme o diagrama de classes citado no capítulo **??**, a aplicação possui algumas entidade, onde entidade é uma tabela presente no banco de dados e um active record contido na pasta app/models.

O Rails possui uma forma bem simples de criar modelos, através de um comando de criação. Seja, por exemplo, a entidade Usuario, tal que seus campos e tipos de campos sejam passados como parâmetros de um comando de criação de modelos:

$rails generate model Usuario nome : string sobrenome : string apelido : string pontos_criador : float pontos_jogador : float username : string senha : string$

A execução desse comando gera os seguintes arquivos:

- Um arquivo de migração de tabela, que cria a tabela usuário no banco de dados
- Uma classe chamada usuario.rb, dentro de app/models, e estende de ActiveRecord
- Arquivos de teste

[3]BSD foi criada pela Universidade de Berkeley e é conhecida por ser uma licença de pouca restrição quando comparada a GNU. Ela permite que o software distribuído sobre a licença seja incorporada a produtos proprietários.

Após a criação do modelo, pode ser rodado o seguinte comando: $rakedb : migrate$

Esse comando tem a função de executar todas as migrações, ou seja, de alterar os dados no banco de dados. Caso seja necessária a modificação de um campo dessa tabela, como por exemplo, ao criar uma associação, pode ser criado um novo arquivo migration especificamente para isso.

### 4.4.3 Relacionamentos

Ao ser analisado o diagrama de classes da aplicação na seção 3.x, pode-se verificar que existem relacionamentos entre as entidades apresentadas. Tais entidades representam relações entre tabelas do banco de dados, as quais devem ser implementadas na camada de modelo da aplicação.

Para relacionar tais modelos, deve-se informar ao Rails o tipo de relacionamento existente. Isso é feito declarando o tipo de relacionamento entre modelos nos active records. Os tipos de relacionamentos tratados são:

- $Belongs_to$: é utilizado quando um modelo tem como atributo o id de outro modelo (em um relacionamento um para muitos ou um para um).
- $has_many$: associação contrária ao $belongs_to$; indica que um determinado modelo tem muitas instancias de outro modelo
- $has_one$: similar ao $has_many$, porém com apenas uma instância (relacionamento um para um)
- $has_and_belongs_to_many$: associação muitos para muitos

As declarações de relacionamento criam alguns métodos de acesso automaticamente. Observe o código abaixo, do modelo de quiz e do modelo de pergunta:

```
#-- classe Quiz
class Quiz < ActiveRecord::Base
  #modojogo: 1- random, 2- ordenate
attr_accessible :titulo, :perguntas_attributes, :modojogo, :maxquesto

has_many :perguntas
end
```

```
#—classe Pergunta
lass Pergunta < ActiveRecord::Base
attr_accessible :conteudo, :respostas_attributes

belongs_to :quiz
has_many :respostas
end
```

Nesses modelos, foi definido que pergunta pertence a quis, e que um quiz possui várias perguntas. Tais definições criam métodos de acesso automaticamente, como:

```
quiz.perguntas #retorna array de perguntas
    pergunta.quiz #retorna objeto Quiz
```

O Rails não verifica automaticamente se a relação declarada funciona. Porém, de acordo com o princípio "convenção sobre configuração", definido na seção 2.x, é esperado que existam chaves estrangeiras do tipo $< modelo >_i$ $d$ nas tabelas que possuam relacionamentos do tipo $belongs_to$. Para tanto, devemos criá-las nas tabelas do banco, utilizando um objeto migration.

Para criar um objeto migration, entre quiz e perguntas, utilizou-se, por exemplo, o seguinte generate do rails:

Rails generate migration AddColumnQuizIdPergunta

Isso gera um arquivo migartion chamado " $< data_hora >$ $add_column_quiz_id_pergunta''$, no diretório /db/migrate. Implementou-se, então, o migration com o campo id:

```
classAddColumnQuizIdPergunta< ActiveRecord::Migration
def up
add_column :Perguntas, :quiz_id, :integer
end

def down
end
end
```

Ao executar o script de execução, rakedb:migrate, será executado um SQL de criação da coluna no banco de dados. É recomendado que as alterações sofridas no banco de dados ao longo do projeto utilizem migration, uma vez que é possível registrar toda a seqüência de alterações no banco e reproduzi-las em outros ambientes.

## 4.5 Rotas e REST

As rotas no rails servem para transformar determinadas requisições URL em chamadas para controles particulares. Algumas dessas rotas, para recursos, são utilizadas para o desenvolvimento da aplicação utilizando REST, em conformidade ao definido na seção 2.x.

Aplicações em Rails são ditas RESTful, associando o o protocolo HTTP às operações de CRUD. Na aplicação, tem-se as seguintes operações disponíveis:

- Get: retorna o valor ou representação do recurso - select
- Post: criação de um novo recurso - insert
- Put: altera o recurso - update
- Delete: remove o recurso - delete

Para acessar o recurso, definimos uma rota de recurso no arquivo routes.rb. Seja o modelo de quiz; define-se o seguinte recurso:

```
# routes.rb
resources :quizzes
```

Para esse recurso, o rails automaticamente cria sete rotas de acesso, a saber:

- GET /quizzes:controller => ' quizzes ', :action => 'index'
- POST /quizzes:controller => ' quizzes ', :action => 'create'
- GET /quizzes /new:controller => ' quizzes ', :action => 'new'
- GET /quizzes /:id:controller => ' quizzes ', :action => 'show'
- PUT /quizzes/:id:controller => 'quizzes ', :action => 'update'
- DELETE /quizzes /:id:controller => ' quizzes ', :action => 'destroy'

- GET /quizzes /:id/edit:controller => ' quizzes ', :action => 'edit'

Para todas as rotas criadas, o Rails também cria os helpers (view do MVC):

- $quizzes_path\# => "/quizzes"$
- $new_quiz_path\# => "/quizzes/new"$
- $edit_quiz_path(3)\# => "/quizzes/3/edit"$

Os resultados dos métodos invocados pelas rotas, por padrão, são em *HyperText Markup Language* (HTML). Entretanto, existem facilitadores no Rails para que esse retorno seja em XML ou JSON. Além disso, podemos passar no corpo de um HTTP uma entrada em XML/JSON, de modo a ser interpretada pelo controller.

Na aplicação, toda a comunicação entre o Web Service (aplicação rails) e o cliente (aplicação mobile) é feita através de Rest utilizando JSON. Isso é possível utilizando um Format, como no exemplo abaixo:

```
# GET /quizzes
  # GET /quizzes.json
def index
    @quizzes = Quiz.all
respond_to do |format|
      format.html # index.html.erb
      format.json { render :json => @quizzes }
end
end
```

O método index é responsável por retornar um array com todos os quizzes. Se a requisição vier por HTML, ou seja, do próprio site, a resposta será em HTML; caso contrario, a resposta será em JSON. Dessa forma, percebemos que a forma como os dados são requisitados é que diferenciam a saída .

## 4.6 Autenticação

Conforme definimos na seção 3.x, um dos requisitos não funcionais é a segurança, ou seja, o sistema deve ser capaz de garantir que os dados de cada usuário estejam protegidos.

A autenticação de um usuário busca verificar a identidade digital do mesmo. A partir da autenticação, o sistema autoriza o acesso ou não a determinados recursos do sistema.

Na aplicação, a autenticação é feita através de tokens. Tokens são chaves criptografadas que identificam uma sessão do usuário. Optamos por disponibiliza-los no momento que o usuário realiza o login. Uma vez que o browser (ou o mobile) obtenha o token, o acesso é garantido até que esse token expire.

A fim de reduzir o tempo de desenvolvimento, utilizamos uma gem chamada Devise, para autenticação e gerência de contas. Essa gem possui doze módulos para gerencia de conta, dos quais utilizamos os seguintes, com as seguntes funções:

- Database authenticable: encripta e armazena uma senha no banco de dados para validar a autenticidade de um usuário enquanto está logado.
- Token Authenticable: usuários permanecem logados enquanto uma token autenticada for valida
- Registerable: permite ao usuário criar contas
- Recoverable: gera uma nova senha para o usuário e o manda instruções para realterar sua senha
- Rememberable: gerencia e limpa tokens para lembrar o usuário de um cookie já salvo
- Trackable: guarda informações de acesso, como o numero de vezes que o usuário logou, o momento do último login e o endereço IP.
- Validatable: Valida a conta através de email/password.

O Devise cria algumas telas (helpers) por padrão. As telas de confirmação, envio de email, gerência de contas, senha e sessão são geradas de maneira básica. As imagens seguintes mostram algumas dessas telas, com o layout modificado pelo bootstrap, que será abordado na sessão seguinte.


Figura 4.1: Tela mostrada quando o usuario tenta acessar recurso e nao está logado


Figura 4.2: Edição de dados do usuário

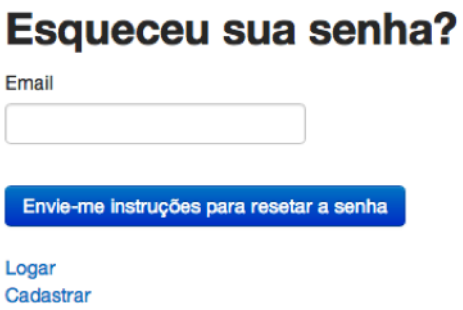


Figura 4.3: Lembrar senha do usuário

Para a validação da sessão no mobile, utilizamos essencialmente manipulação

por tokens. No momento que o usuário abre a aplicação, é verificado se o token é válido: caso não seja, uma tela pedindo nome de usuário e senha é mostrada. Após o usuário fornecer os dados, é enviada uma requisição em JSON para `/api/v1/token_controller`, que verifica se os dados estão corretos e se o usuáriorealmente existe. Se tudo ocorrer corretamente, um token é retornado da requisição; caso contrario, uma mensagem contendo o erro é retornada.


Figura 4.4: Login no mobile (iOS)

## 4.7 Front-End

No modelo MVC, a parte de view (visão) é responsável pelas interações com o usuário, recebendo as entradas e renderizando a saída. Os tipos de arquivos de fronteira mais comuns são o HTML, ERB e o eRuby. Em nossa aplicação, utilizamos em quase todas as páginas a extensão ERB.

Os controllers, por sua vez, são responsáveis por receber a ação de uma View e executar alguma lógica ligada a algum modelo. Basicamente, a função do controller é atender a requisições entre modelo e visão, fazendo toda a tradução necessária

entre as camadas.

Um controller pode ser criado através de um comando generate, como a seguir: Rails generate controller quizzes

Esse comando irá criar uma classe chamada *quizzes_controller.rb* e uma pasta chamada quizzes, dentro da pasta views. A ligação entre as camadas é feita automaticamente, devido ao conceito de "convenção sobre configuração".

Nesse controller criado, foram colocados métodos de manipulação de dados (RESTful). Na parte de visão, colocamos alguns arquivos para manipular quizzes.

Veja, a seguir, o código da página index.html.erb. que tem como função listar quizzes e navegar para tela de criação:

```
<h1>Quizzes</h1>

<table class="table table-hover">
        <thead>
<tr>
<th>#</th>
<th>Título</th>
<th>Descrição</th>
        <th>Criador</th>
        <th>#Perguntas</th>
</tr>
</thead>
<tbody>
        <% @quizzes.each do |quiz| %>
                <tr>
                        <td><%= quiz.id %></td>
                        <td><%= link_to quiz.titulo, quiz %></td>
                        <td><%= quiz.descricao %></td>
                        <td><%= link_to quiz.user.username, quiz.user
                        <td><%= quiz.perguntas.size %></td>
                </tr>
        <% end %>
</tbody>
</table>
```

```
<%= link_to new_quiz_path do %>
        <button class="btn btn-large btn-block btn-primary" type="but
<% end %>
```

Nesse código, é possível observar que existe código Ruby inserido entre os caracteres "<%" e "%>". Esses caracteres representam um escape do html para o Ruby. Quando colocado com um sinal de igual, "<%=", o resultado da linha executada é impressa na tela final HTML apresentada ao usuário.

É importante observar que todos os atributos de classe do controller, representados por um @, como em @quizzes, estão sobre o mesmo escopo.

### 4.7.1 Bootstrap

"Elegante, intuitiva e poderoso framework de front-end para desenvolvimento web rápido e fácil" - com essa descrição, o Bootstrap é apresentado em seu site oficial (**??**). Lançado em Agosto de 2011 pelo Twitter, é atualmente o projeto mais popular do GitHub (**??**).

O Bootstrap é uma coleção livre de ferramentas para desenvolver sites e aplicações web. Basicamente, contém templates HTML, JavaScript e *Cascading Style Sheets* (CSS) para tipografia, formas, botões, gráficos, navegações, etc. Na aplicação, utilizamos o Bootstrap para personalização de botões, tabelas, divisão da página em duas seções e menu.


Figura 4.5: Exemplo de controle bootstrap

Uma das funcionalidades mais aclamadas do Bootstrap é a adequação da tela para resoluções menores: quando colocado em telas de pouca largura, tipicamente em smartphones, a página se adequa para ser apresentada em um layout mais amigável para dispositivos móveis, como abaixo:


Figura 4.6: Tela desktop extendida


Figura 4.7: Tela mobile com menu suprimido


Figura 4.8: Tela mobile com menu extendido

## 4.8 Deploy

Para que uma aplicação rails execute, é necessário um servidor de aplicação. Em ambiente de desenvolvimento, utilizamos o Webrick, que é um servidor já embutido em todas as aplicações. Para executá-lo, basta abrir o terminal, navegar até o diretório raiz da aplicação e executar o seguinte comando:

rails server

Caso uma porta diferente da porta padrão seja necessária (3000), pode ser passado o parâmetro -p xxxx, onde xxxx é a porta desejada.

O nosso ambiente de produção está hospedado como serviço no Heroku, sobre o runtime stack "Celadon Cedar". Para ativar, foi necessário fazer o cadastro, instalar o Heroku Toolbelt, efetuar login via terminal e fazer deploy. Mais detalhes podem ser encontrados no QuickStart do Heroku (**??**).

A submissão de arquivos para o servidor do Heroku é feita através do controlador de versão Git. Para submeter, deve-se comitar o código e realizar um git push para o servidor. Caso seja necessário executar algum comando, como rake de migrations, usa-se o "run", conforme a seguir:

heroku run rake db:migrate

Capítulo 5

# Comparativo Android x iOS

## 5.1 Ambientação

iOS Para implementação nativa de aplicações, utilizamos a IDE gratuita chamada Xcode. Desenvolvida pela Apple Inc., funciona apenas sobre o sistema operacional Mac OS X. Por padrão, já vem com suporte ao Objective-C, linguagem de programação utilizada para desenvolvimento de aplicativos nativos no Mac OS X e no iOS. Atualmente, o Xcode está na versão 4.5.2 e pode ser baixado via Mac App Store, de forma gratuita para usuários do Mac OS X Lion e OS X Mountain Lion `https://itunes.apple.com/us/app/xcode/id497799835`. Para desenvolvimento iOS, é necessário possuir o iOS SDK, que pode ser baixado internamente a ferramenta Xcode. Entretanto, para realização de testes no dispositivo, é necessário a posse da licença de desenvolvimento.

Android Para o desenvolvimento de aplicações nativas em Android, necessitamos de um ambiente que possua Android SDK instalado. É possível, por exemplo, gerar aplicativos utilizando apenas um editor de texto, como Notepad ou Gedit. Entretanto, a Google recomenda o uso do Eclipse com ADT Plugin (link).

### 5.1.1 IDEs

## 5.2 Arquitetura

### 5.2.1 Arquivos gerados

## 5.3 Controles básicos

### 5.3.1 Mostrando textos

iOS Para exibir textos sem interação direta com o usuário, utilizamos uma instância da classe UILabel. Essa classe possui algumas propriedades para modificação de aspectos, a saber:

- text: o texto mostrado no campo;
- textColor: cor do texto;
- numberOfLines: número máximo de linhas suportadas;
- font: fonte utilizada pelo texto, instância de UIFont.

Android

De maneira similar ao iOS, o Android possui um controle específico para mostrar textos na tela, chamado TextView. Um TextView pode ser definido no layout XML ou no próprio Java. Assim como no iOS, possui uma propriedade do tipo String chamada "text", a qual representa o texto apresentado. Possui também algumas propriedades, a saber: [as propriedades legais] Uma diferença importante entre o UILabel e o TextView é que o último aceita interações com o usuário; podemos colocar eventos de interação com o usuário. Essa funcionalidade costuma ser usada com textos que representam links (URLs).

### 5.3.2 Inserindo textos

iOS Para entrada de textos, utilizamos uma istância de UITextField. A manipulação do texto é feita através do disparo de uma ação para um "target" quando o usuário pressiona o botão "return" do teclado. Essa classe normalmente é associada a um UITextFieldDelegate, o qual fornece métodos adicionais de decisão. Quando o usuário toca em um textfield, esse controle torna-se o "first responder" e

invoca o aparecimento do teclado para o sistema. O teclado deve ser configurado, pelo desenvolvedor, para desaparecer quando o botão de return for pressionado. Isso deve ser feito através da mensagem “resignFirstResponder”, a ser enviada para o textfield. Para que o textfield não fique “escondido” na tela, embaixo do teclado, cabe ao desenvolvedor mover tela de maneira conveniente, de forma a aparecer o conteúdo. A aparência do teclado pode ser configurada utilizando o protocolo UITextInputTraits. Existem, entretanto, alguns tipos de teclado a serem definidos por padrão, como o ASCII, Number, Url, Email, entre outros.

Android

Para a entrada de texto do usuário, o Android fornece o controle chamado EditText. Assim como o UITextField, esse controle possui uma propriedade chamada "text", que representa o texto mostrado no controle. Esse texto pode ser tanto a entrada do usuário como também um texto configurado via programação pelo aplicativo. O EditText possui algumas propriedades que usamos na aplicação, como as seguintes: [propriedades legais] Para cada EditText, podemos definir o tipo de teclado que será exibido ao interagir com o usuário, como teclados de telefone, numeros e de letras. O EditText possui duas grandes diferenças em relação ao UITextField, a saber:

- O teclado retorna automaticamente quando terminada a edição;
- Ao entrar em modo de edição, o EditText faz com que a tela do aplicativo desloque-se, caso seja necessário para aparecer o conteúdo do controle.

### 5.3.3 botões e eventos

iOS

Um botão é representando por uma instância da classe UIButton. Os botões interceptam eventos de toque e enviam mensagens para um target pré-definido quando tocados. Métodos para configurar os targets e ações são herdados de UIControl. Possuem título, imagem e outras propriedades de aparência.

Os botões respondem a algumas ações, como touch drag, touch down, touch up, entre outros. Para associar um método a um evento, criamos uma IBAction (método) e, ao evento, indicamos tal IBAction. Isso pode ser feito facilmente no Interface Builder/Xcode “ligando” o botão ao código correspondente (file’s owner desse botão), conforme os passos abaixo:

**i.** Arrastar, com o botão direito, o botão ao código do file's owner

**ii.** Nomear um IBAction e associar um evento

Android

Os botões no Android pertencem a classe Button. Existem duas formas de inserir eventos nos botões, a saber:

- via Listener Java: podemos implementar o OnClickListener na Activity ou em alguma classe anônima dentro da Activity, de modo que ao disparar o evento de click, o método onClick da interface seja chamado.
- via propriedade onClick no XML: os botões possuem uma propriedade chamada onClick, que recebem uma String como valor. Esse valor é o nome do método da Activity onde o layout contendo esse botão foi inflado. Em nosso projeto, utilizamos somente onClick como propriedade XML, uma vez que toda a parte de layout foi implementada via XML.

## 5.4 Criando listas

iOS

Listas em iOS são feitas utilizando instâncias de UITableView. Essa classe, por sua vez, estende de UIScrollView, que habilita a rolagem vertical (e apenas a vertical). Internamente, cada linha (célula) é representada por um objeto de UITableViewCell, que são totalmente configuráveis. Esse controle normalmente é associado a um UINavigationController: quando uma célula é tocada, é feito um push de um novo UIViewController, detalhando a célula. Table views possuem dois estilos, a saber: UITableViewStylePlain e UITableViewGrouped. Uma vez criado o controle, não é possível mudar o estilo. No estilo Plain, as seções de header/footer flutuam nas bordas do conteúdo. Nesse estilo, pode haver um índice variando de A - Z, que facilita a navegação vertical. No estilo Grouped, uma cor padrão é definida para o fundo da view e das células. Nesse estilo, podem ser criadas várias listas, de formas agrupadas. Nesse caso, não podem ter índice. Muitos métodos utilizam o objeto do tipo NSIndexPath como parâmetro e retornam valores. Esse objeto representa o índice da linha atual e da seção atual. Um objeto do tipo Table View necessita de um objeto que atue como fonte de dados (data source)

e um objeto que atue como delegate. Normalmente, utilizamos os protocolos UITableViewDataSource e UITableViewDelegate no UIController no qual o table view esteja inserido. O data source fornece informação que o UITableView precisa para construir tabelas e o delegate fornece as células e executa alguns outros métodos de manipulação. A fim de criar uma table view básica, precisamos implementar, no mínimo, os seguintes métodos data source:

**i.** Número de seções - retorna o número de seções para essa table view

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
```

**ii.** Número de rows (linhas): retorna o número de linhas para cada uma das seções

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSe
(NSInteger)section
```

**iii.** cell for rows at index: define uma célula para cada index (par linha/seção) da tabela

```
-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIn
\end{enumerate}
```

Além desses métodos, é comum implementar um método delegate que responda a cada célula tocada, como a seguir:

I. did select row at index: informa o par (linha/seção) que foi selecionado. A partir do index path, pode-se recuperar a célula selecionada dentro da table view

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection
(NSInteger)section
```

Para a tela de escolha dos quizzes (GameMenu), implementamos uma table view simples, da seguinte maneira:

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
return1;
}
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
//conta no array de quizzes a quantidade de elementos
return [self.quizzescount];
```

```
}
-(UITableViewCell*)tableView:(UITableView *)tableView cellForRowAtInd
//reaproveita ou cria uma célula
UITableViewCell* cell = [tableView dequeueReusableCellWithIdentifier:
if (cell == nil){
cell = [[UITableViewCellalloc] init];
}

//personalização da célula
Quiz* q = [self.quizzesobjectAtIndex:indexPath.row];
cell.textLabel.text = [q titulo];
return cell;
}
//disparado quando um dos quizzes for selecionado
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NS
Quiz* q = [self.quizzesobjectAtIndex:indexPath.row];
GameViewController* gq = [[GameViewControlleralloc] initWithNibName:@
gq.quiz = q;
NSLog(@"quiz.content: %@",q.titulo);
NSLog(@"count perguntas: %d", q.perguntas.count);
[self.navigationControllerpushViewController:gq animated:YES];
}
```

Android As listas em Android são instâncias da classe widget ListView e, assim como no iOS, já possui rolagem vertical implementada. Os itens da lista são inseridos por outra classe controladora, chamada Adapter. O Adapter é uma classe que provê acesso aos itens que contém a informação, como um array de strings. Além disso, o Adapter é responsável por criar o layout de cada célula. Para a lista simples mostrada em GameMenu, foi implementado uma lista com células padrão da seguinte maneira:

```
    private void carregarLista() {

      ArrayAdapter arrayAdapter = new ArrayAdapter(this, android.R.lay
   ListView listView = (ListView)findViewById(R.id.lista_quizzes);
listView.setAdapter(arrayAdapter);
listView.setOnItemClickListener(this);
```

```
}
```

Nesse trecho de código, quizzes é um ArrayList de quizzes. ArrayAdapter e ListView são classes padrão do Android. Os eventos de clique de célula estão associados a classe que implementa a interface OnItemClickListener. Tal interface exige a implementação do método onClickListener. Foi utilizado, no iQuizzer, esse listener na classe GameMenu, tendo sido implementado o método da seguinte maneira:

```
        @Override
public void onItemClick(AdapterView<?> adapterView, View view, int po
Quiz quiz = quizzes.get(position);
  Intent i = new Intent(getApplicationContext(), GameActivity.class);
  i.putExtra("quiz",quiz);
  startActivity(i);
}
```

## 5.5 Personalizando linhas de uma lista

– retirado iOS Android

## 5.6 Acesso a dados

### 5.6.1 SQLite

O SQLite é uma biblioteca implementada em C, de domínio público, que representa um banco de dados SQL. Tem como características fundamentais ser contido em si mesmo, livre de servidores, sem configuração e transacional. (`http://www.sqlite.org/about.html`). Tanto o iOS quanto o Android possuem abstrações para representar objetos da tabela (active records), conexões e o próprio banco.

iOS Existem duas maneiras de se trabalhar com SQLite em iOS: acesso nativo através da biblioteca SQLite.h e Core Data Framework. No acesso nativo, a programação utilizada é a mesma para aplicações nativas em C, ou seja, não há particularidades entre utilizar o SQLite dentro ou fora do iOS. Já em Core Data, a programação é em Objective-C e existe uma camada de abstração do banco de dados. Em nossa aplicação, optamos por usar o Core Data devido à facilidade de fazer a modelagem utilizando o xcdatamodel. O xcdatamodel é uma ferramenta

facilitadora, integrada ao Xcode, na qual o desenvolvedor pode "desenhar" as tabelas e criar ligações, chamadas "relationships". Cada uma das tabelas e relações possuem propriedades, que são facilmente configuradas.

Existem três classes principais de abstração de banco de dados, a saber:

- Managed Object Model: é a classe que contém as definições de cada objeto (entidades, active record);
- Persistent store coordinator: é a conexão do banco; são configurados os nomes e a localização da base de dados;
- Managed Object Context: classe responsável pelas operações básicas de manipulação: inserir objetos, deletar objetos, etc.

No iQuizzer para iOS, criamos uma classe chamada DAO e implementamos métodos de CRUD, como pesquisar, inserir, modificar e deletar. Para cada uma das entidades, criamos uma classe de DAO estendida, como QuizDAO ou PerguntaDAO, onde cada uma dessas realiza operações de crud mais especificas e voltadas para as situações que ocorrem na aplicação. Nessas classes de DAO, fazemos manipulação de "managed object contexto" e "persistente store coordinator". Para cada uma das entidades, criamos um managed object model, estendendo da classe nativa NSManagedObject. É possível criar tais classes com os atributos e relações pré-configuradas selecionando o template "NSManagedObject subclass" e selecionando o data model desejado.

Android

O framework Android possui um pacote chamado android.database.sqlite, que fornece todas as classes necessárias para o gerenciamento do banco de dados privado a cada aplicação. Por padrão, o Android vêm com a versão 3.4.0 do SQLite. Em nossa aplicação, utilizamos as seguintes classes desse pacote:

- SQLiteOpenHelper: Essa classe possui métodos para abrir o banco, como onCreate(SQLiteDatabase) e onUpgrade(SQLiteDatabase, int, int), que abrem, criam e atualizam o banco caso necessário.
- SQLiteDatabase: possui métodos para gerenciar o banco SQLite. Com essa classe, foram feitas as interações com as entidades do banco, utilizando essencialmente o método execSQL(String) para entrada e manutenção de tuplas.

- Cursor: classe de manipulação de resultados de uma consulta.

### 5.6.2 Preferências

Existem casos onde a complexidade da informação a ser armazenada é pequena, como pares de chave-valor. Em casos onde temos uma informação simples a ser armazenada, utilizamos a memória de preferências para gravar tais valores. Basicamente, essas classes de preferências atuam como um HashTable, que armazena uma estrutura de chave e valor para tipos primitivos, e os valores armazenados estarão no escopo da aplicação mesmo que a aplicação seja encerrada ou o dispositivo desligado. O funcionamento básico dessa funcionalidade é similar no iOS e Android: uma chave (String) é escolhida para "batizar" a informação a ser armazenada. Ao momento de inserir/modificar, chamamos de "set". Para recuperar o valor, invocamos um método "get", passando o nome da variável que fora batizada como parâmetro. Em nosso aplicativo, utilizamos as memórias de preferências para armazenar o token. Quando a aplicação começa, verifica se há um valor de token válido armazenado. Em caso positivo, a aplicação inicia na tela de menu; caso contrário, é mostrada a tela de login.

iOS No iOS, a classe que representa a memória de preferências é chamada de NSUserDefaults. Para ler ou escrever, necessitamos de uma instância de NSUserDefaults - em nossa aplicação, standardUserDefaults. Com a instância, e de posse do valor da chave, podemos escrever os seguintes métodos de acesso:

**i.** Instância padrão:

**ii.** Configurando valor para token:

```
[defaults setObject:token forKey:@"token"]; //escrita de token
```

**iii.** Recuperando o valor de token:

```
[defaults objectForKey:@"token"];
```

Android

No Android, a classe que representa a memória de preferências é chamada deSharedPreferences. De maneira similar ao iOS, temos um método para atribuir valor e outro para recuperar; entretanto, no momento da recuperação, é passado

um valor padrão a ser retornado caso não exista o valor procurado. Além disso, é preciso comitar a memória depois de adicionar um determinado valor. A atribuição e a recuperação são feitas da seguinte maneira:

**i.** Instância padrão:

SharedPreferencespreferences = PreferenceManager.getDefaultSharedPreferences(context);
SharedPreferences.Editor editor = preferences.edit();

**ii.** Configurando valor para token:

```
editor.putString("token", token);
editor.commit();
```

**iii.** Recuperando o valor de token:

```
String token = preferences.getString("token", "");
```

## 5.7 Parser JSON

A fim de manipular objetos JSON, tanto na formação quanto na interpretação, existem bibliotecas nativas incorporadas aos frameworks nativos do iOS e do Android. Tais bibliotecas trabalham de maneira similar, utilizando o conceito de chave e valor. Em nossa aplicação, todas as mensagens trocadas entre mobile e servidor utilizam JSON, sendo de vital importância a compreensão dessa funcionalidade. Para baixar quizzes, é recebido o seguinte JSON do servidor, no corpo do HTTP:

```
{
    "quiz": {
        "created_at": "2012-11-05T16:43:14Z",
"descricao": "Perguntas sobre fatos marcantes que ocorreram no ano de
        "id": 3,
        "maxquestoes": 5,
        "modojogo": 1,
        "titulo": "Retrospectiva 2012",
"updated_at": "2012-12-28T10:18:01Z",
        "user_id": 1,
"perguntas": [.... ]
```

```
}
}
```

iOS A partir do iOS 5, podemos utilizar a classe NSJSONSerialization para criar e interpretar JSON. Essa classe trabalha com objetos comuns de representação de dados, como NSString, NSNumber, NSArray e NSDictionary, não necessitando de No método de baixar quiz do servidor, utilizamos a seguinte conversão entre o objeto JSON (um NSData contendo o corpo do HTTP) e um objeto Quiz:

```
NSError* error;

    NSDictionary* jsonObj = [NSJSONSerialization JSONObjectWithData:j

    NSDictionary* jsonQuiz = [jsonObj objectForKey:@"quiz"];

//Instanciando quiz a partir de uma entidade
    Quiz* quiz = [[Quiz alloc] initWithEntity:entityDescription insert

quiz.titulo = [jsonQuiz objectForKey:@"titulo"];
quiz.descricao = [jsonQuiz objectForKey:@"descricao"];
quiz.maxquestoes = [jsonQuiz objectForKey:@"maxquestoes"];
quiz.modojogo = [jsonQuiz objectForKey:@"modojogo"];
quiz.index = [jsonQuiz objectForKey:@"id"];
```

Ou seja, utilizamos apenas uma conversão entre o JSON e um NSDictionary. Caso a raiz do JSON fosse um array, a única modificação seria que o objeto a ser retornado da conversão seria um NSArray. Para criar um objeto JSON a partir de um NSDictionary ou NSArray, utilizamos o método contrário ao que utilizamos na interpretação do JSON, conforme abaixo:

```
NSArray* objects = [[NSArray alloc] initWithObjects:jogo.dia, jogo.ho
NSArray* keys = [[NSArray alloc] initWithObjects:@"dia",@"hora",@"por

    NSMutableDictionary* jsonDict = [[NSMutableDictionary alloc] init

NSData* jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict o
```

Android

No Android, existe uma implementação da biblioteca official disponível em (www.json.org) interna ao framework. Ou seja, a maneira de se trabalhar com JSON é a mesma de aplicações java que utilizam tal biblioteca. No método de baixar quiz do servidor, utilizamos a seguinte conversão entre o objeto JSON (umaString contendo o corpo do HTTP) e um objeto Quiz:

```
JSONObject jsonObj = new JSONObject(jsonData);
                    JSONObject jsonQuiz = jsonObj.getJSONObject("quiz");

                    Quiz quiz =  new Quiz(jsonQuiz.getInt("id"), jsonQuiz
```

Verificamos que a maneira de interpretar JSON é bem parecida no iOS e no Android. A diferença notável é que, enquanto o iOS trabalha com objetos "nativos" como NSDictionary e NSArray, o Android trabalho com wrappers JSONObject e JSONArray. Na criação de um JSON, como no método de criar jogo para enviar o resultado ao servidor,fora utilizada a seguinte conversão:

```
JSONObject jsonObject = new JSONObject();
  try{
          jsonObject.put("dia", jogo.getDia());
          jsonObject.put("hora", jogo.getHora());
          jsonObject.put("pontos", jogo.getPontos());
          jsonObject.put("resultados_attributes", jsonR
          jsonObject.put("user_id", usuario_id);
  } catch (Exception e ){
          e.printStackTrace();
  }
  return jsonObject.toString();
```

Novamente podemos observar que a maior diferença é relativa ao wrapper JSONObject e JSONArray, presentes apenas na implementação Android.

## 5.8 Conexão HTTP

Conforme já citado, toda a troca de mensagens entre o servidor e nossa aplicação móvel é feita através do protocolo HTTP. Dentro do corpo da mensagem, enviamos ou recebemos um JSON, contendo normalmente um recurso (como uma instância

de Quiz ou Pergunta, por exemplo). O protocolo HTTP possui alguns campos em seu cabeçalho que são especialmente úteis para nossa aplicação. São eles:

- Method: nesse campo definimos o tipo de método HTTP que será executado. Esse campo é essecialmente importante para aplicações REST, uma vez que usa os mesmos (GET, POST, PUT e DELETE) para realizar as operações de CRUD.

- Content-Type: nesse campo é definido o tipo de arquivo que estará sendo enviado. Para que a aplicação Rails saiba que está sendo enviada uma requisição via mobile com JSON interno, utilizamos o content-type "JSON"para diferenciar. Caso seja uma requisição web normal, vinda do browser, o content-type é "text/plain".

Existem duas formas de tratar as conexões HTTP: sincronamente e assincronamente. Por uma questão de tempo e aproveitamento de código, optamos pela maneira síncrona; entretanto, acreditamos que a maneira assíncrona seja mais bem aceita pelo usuário, uma vez que não interrompe a execução da aplicação e é apontada como trabalho futuro dessa aplicação, na seção 7.3. Em nossa aplicação, criamos uma classe chamada WebService, que possui métodos de comunicação com o servidor web através do protocolo HTTP.

iOS

A implementação de requisições HTTP é feita utilizando as seguintes classes:

- NSURL: representa a url que será enviada à requisição;

- NSURLRequest: representa a requisição HTTP. Na instância dessa classe, definimos o método HTTP desejado, o content-type e o corpo da mensagem;

- NSURLConnection: representa a conexão entre o dispositivo e o servidor, possuindo uma url e um request. Além disso, indica a classe delegate da requisição, ou seja, a classe que possui os métodos necessários para tratar as respostas dessa requisição

Na classe WebService, foram criados os métodos get e RESTCommand. No método RESTCommand, além de configurarmos a url, o cabeçalho e o corpo do HTTP, definimos um delegate para o objeto NSURLConnection. Esse delegate é

responsável por tratar a resposta da requisição, através dos métodos didReceiveData e connectionDidFinishLoading.

Android

Para utilizar a maneira síncrona, deve-se primeiramente modificar a política de threads padrão do Android. Isso foi feito adicionando o seguinte código no método onCreate da activity principal:

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.
StrictMode.setThreadPolicy(policy);
```

Para realizar requisições HTTP, utilizamos instâncias das seguintes classes:

- URL: representa a url que será enviada à requisição;
- HttpClient: representa o cliente (alvo) da requisição;
- HttpURLConnecction: representa a conexão e configura cabeçalhos HTTP;
- OutputStream e BufferedReader: gerenciam os bytes enviados e recebidos durante a transmissão.

De maneira análoga ao iOS, foram criados os métodos get e RESTCommand, de forma a fazer a comunicação síncrona.

Capítulo 6

# Resultados

Capítulo 7

# Conclusão

## 7.1 Contribuições

Este trabalho trouxe duas contribuições importantes: a primeira, referente ao aplicativo iQuizzer; a segunda, referente a documentação de configuração e implementação de arquitetura de aplicativo móvel utilizando web services.

O aplicativo *iQuizzer*, disponibilizado nas versões web e mobile, estimula a busca por conhecimento de um determinado assunto, tanto para quem está criando o quiz quanto para quem está jogando. Além disso, permite que enquetes sejam feitas de maneira rápida, com resultados sendo avaliados em uma plataforma web.

A documentação gerada fornece uma base para muitas das aplicações que estão surgindo utilizando a abordagem de web service com computação móvel. Tal abordagem vem sendo utilizadas por um grande número de *Startups*, as quais necessitam, essencialmente, desenvolver e avaliar seus produtos de maneira rápida, mesmo que o produto ainda esteja em fase de desenvolvimento.

## 7.2 Limitações

A aplicação *iQuizzer* tinha como um dos objetivos ser a validação de uma idéia onde quizzes são criados e acompanhados na web e jogados via mobile. Como o intuito era a validação, algumas funcionalidades não foram implementadas por não interferirem diretamente na validação da idéia.

Algumas das funcionalidades não implementadas são limitações do projeto, como: a não atualização dos quizzes depois de baixados, mesmo que modificados

na web; a não adoção de conexões assíncronas; a falta de integração com as redes sociais; a falta de criptografia na troca de mensagens.

No que concerne as plataformas de desenvolvimento mobile, existem duas limitações. O aplicativo desenvolvido em *iOS* não funciona em versões inferiores ao *iOS 5.0*, devido ao uso de bibliotecas exclusivas dessa versão. De maneira similar, o aplicativo desenvolvido em *Android* não funciona em versões inferiores ao *Android 2.3 Gingerbread*.

## 7.3 Trabalhos futuros

A fim de lançar a plataforma *iQuizzer* no mercado, de maneira escalável e competitiva, algumas funcionalidades deveriam ser implementadas:

- Integração com as redes sociais: o módulo web poderia ser disponível como um aplicativo para *Facebook*, onde o controle de usuários seria feito pela própria conta do usuário na rede. Além disso, as ações do usuário, tanto na parte web como na parte mobile, podem corresponder a ações no Facebook, de modo a divulgar a plataforma de forma viralizada.
- Segurança: toda a troca de mensagens pode ser feita utilizando protocolos seguros, como *Secure Shell* (SSH). Dessa forma, poderia garantir a inviabilidade da informação trocada.
- Elementos de rede social: ações como "curtir", "comentar" e "seguir usuário" podem ser colocadas dentro da plataforma, de modo a aumentar a experiência do usuário em relação a outros usuários.
- Disponibilização do jogo para outras plataformas: a função de jogar um quiz pode ser disponibilizada na web, no *Windows Phone*, entre outros.
- Conexões assíncronas, de modo a otimizar a experiência do usuário em relação a usabilidade.
- Melhorias no layout: no estado atual, a aplicação não possui uma identidade visual, o que é de vital importância para o sucesso comercial do aplicativo.
- Notificações e controles de atualização de quizzes.

- Monetização, com propagandas ou venda de quizzes: pode-se monetizar a aplicação colocando banners, tanto na web como no mobile. Poderia, também, ser criado um mini-sistema de venda de quizzes, onde os criadores de cada quiz lucrassem com a compra de quizzes pelos jogadores.