UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ESTEBAN DAMIÁN BARREA

ESTUDO DE CASO PARA O DESENVOLVIMENTO DE UM SISTEMA WEB BASEADO EM VRAPTOR E ANDROID

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2016

ESTEBAN DAMIÁN BARREA

ESTUDO DE CASO PARA O DESENVOLVIMENTO DE UM SISTEMA WEB BASEADO EM VRAPTOR E ANDROID

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Marcio Angelo Matté.

MEDIANEIRA

2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Desenvolvimento de sistema para bancas de TCC

Por

Esteban Damián Barrea

Este Trabalho de Diplomação (TD) foi apresentado às 13:00 hs do dia 09 de Junho de 2016 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Manutenção Industrial, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Esp. Marcio Angelo Matté
UTFPR – *Campus* Medianeira
(Orientador)

Prof. MSc. Ricardo Sobjak
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Esp. Valter Rodrigo Ekert
UTFPR – *Campus* Medianeira
(Convidado)

Prof. MSc. Jorge Aikes Junior
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

BARREA, Esteban Damián. ESTUDO DE CASO PARA O DESENVOLVIMENTO DE UM SISTEMA WEB BASEADO EM VRAPTOR E ANDROID. Trabalho de conclusão de curso – Curso Superior De Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Medianeira, 2016.

A Universidade Tecnológica do Paraná (UTFPR) do campus de Medianeira nas áreas de Ciências da Computação (CC) e Tecnologia em Análise e Desenvolvimento de Sistemas (TADS) utiliza como método de avaliação final para o aluno receber seu diploma o trabalho de conclusão de curso (TCC) o qual e feito a partir do conhecimento adquirido por este durante o seu período de estudos na universidade e avaliado por uma banca formada pelos seus professores.

O sistema de avaliação de bancas de TCC é um projeto que surgiu da necessidade de um sistema que simplifica os métodos de avaliação e armazenamento das notas dos professores durante a banca. O sistema conta com uma versão *Web* e que interage com uma versão para *Android*, a versão *Web* tem a finalidade de gerir o sistema, mais também conta com uma tela de cadastro de notas, já a versão para *Android* foi planejada somente para o cadastro de notas da banca.

Palavras-chave: Bancas de TCC, Java, Frameworks, REST.

RESUMO EM LÍNGUA ESTRANGEIRA

BARREA, Esteban Damián. CASE STUDY TO BUILDING AN WEB SYSTEM BASED IN VRAPTOR AND ANDROID. Completion of course work - Degree Technology Analysis and Systems Development, Federal Technological University of Paraná. Medianeira 2016.

The Technological University of Paraná (UTFPR) campus of Medianeira in the areas of Computer Science (CC) and Technology Analyses and Systems Development (TADS) uses as a final evaluation method for the student to receive his diploma course conclusion work (TCC) which made and from the knowledge acquired by this during their period of study at the university and evaluated by a panel formed by their teachers.

TCC stands evaluation system is a project that arose from the need for a system that simplifies the methods of evaluation and storage of the notes of the teachers during the bank. The system has a web version and interacting with a version for Android, the web version is intended to manage the system, plus also has a note registration screen, as the Android version is designed only for the registration bank notes.

Keywords: Bank of TCC, Web, Android, Frameworks.

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
CC	Ciências da Computação
CRUD	Create Remove Update Delete
CSS	Cascading Style Sheet
DAO	Data Access Object
HTML	Hipertext Markup Language
IDE	Integrated Development Environment
JavaEE	Java Enterprise Edition
JSP	Java Server Pages
MER	Modelo Entidade Relacionamento
MVC	Model View Controller
SGBD	Sistema Gerenciador de Banco de Dados
TADS	Tecnologia em Análise e Desenvolvimento de Sistemas
TCC	Trabalho de Conclusão de Curso
REST	Respresentation State Transfer
UTFPR	Universidade Tecnológica Federal de Paraná

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de um sistema empresarial tem por objetivo o alinhamento de diversos setores, ganho de tempo durante a rotina diária da empresa, a redução de perca de dados, etc. Um sistema pode abarcar vários dispositivos, com isto os sistemas *Web* vêm ganhando fama, já que não é necessário o programa estar instalado na máquina, não é necessário o desenvolvimento para vários dispositivos, já que o mesmo funciona via navegador, e caso seja necessária a utilização de uma versão instalada em computadores, celulares e afins, se pode utilizar alguns *Frameworks* durante o desenvolvimento do sistema *Web* que façam conexões/transações, como o *Respresentation State Transfer* (REST), com estes sistemas específicos para dispositivos.

Hoje com o avanço tecnológico temos celulares capazes de auxiliar as nossas tarefas diárias e até ajudar no trabalho. Estes celulares contam com sistemas operacionais inclusos neles, o que facilita a criação de aplicativos e estes possam ter diferentes utilidades.

O *Android*, sistema operacional para *Smartphones* pertencente a Google, é utilizado na maioria dos celulares hoje em dia, ele abarca 46,87% dos celulares (NETMARKSHARE, 2016).

As universidades e faculdades utilizam como método de avaliação final para o aluno receber seu diploma o trabalho de conclusão de curso (TCC) o qual e feito a partir do conhecimento adquirido por este durante o seu período de estudos na universidade e avaliado por uma banca formada pelos seus professores.

Neste trabalho foi projetado e desenvolvido um sistema para a automatização destas bancas de TCC, com o uso de tecnologias *Java Enterprise Edition* (JavaEE) e VRaptor4 com a utilização de REST para a comunicação entre o sistema *Web* e *Mobile* para a plataforma *Android*.

1.1 OBJETIVO GERAL

Utilizar *Frameworks Web* e dispositivos móveis no desenvolvimento de um sistema para avaliação de trabalho de conclusão de curso.

1.2 OBJETIVOS ESPECÍFICOS

- Estudo de *Frameworks* para JavaEE versão *Web* com VRaptor 4;
- Estudo de utilização de métodos REST via VRaptor 4 para integração *móbile*;
- Estudo de *Frameworks* em Java para desenvolvimento *Android*;
- Desenvolvimento do Sistema em versão *Web*;
- Desenvolvimento do Sistema em versão *Android*.

1.3 JUSTIFICATIVA

A abordagem deste trabalho apresenta os resultados de um sistema *Web* desenvolvido com base em JavaEE usando o *Framework* VRaptor e utilizando as tecnologias REST para a comunicação entre o sistema *Web* e a aplicação para *Android*, visando facilitar as avaliações das bancas de TCC.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está dividido em cinco capítulos. No capítulo um é feita uma introdução sobre o panorama do tema a ser estudado, bem como, os objetivos específicos, objetivo geral e justificativa.

No capítulo dois é feito o embasamento teórico sobre assuntos que circundam o tema principal. Neste capítulo o foco é mais geral e não tem como objetivo aprofundar em cada item nele contido.

O capítulo três contém o estudo aplicado sobre o tema central do trabalho.

Já os capítulos quatro e cinco apresentam os resultados e discussões e ainda o desfecho com a conclusão do trabalho, respectivamente.

2 REFERENCIAL TEÓRICO

No percurso deste capítulo será apresentado processo de avaliação de bancas de TCC, necessários para o entendimento da necessidade dos professores e desenvolvimento do sistema.

2.1 A INTERNET

Durante muito tempo as informações foram passadas de boca a boca ou por meio de cartas, até o surgimento da Internet como é conhecida hoje, mas no seu começo não era parecida comparada a de hoje, o seu começo até onde é conhecido foi após o lançamento de primeiro satélite artificial pela União Soviética em 1958, com isto a União Soviética representou uma grande ameaça para os Estados Unidos os quais pensavam que a mesma poderia ataca-los a qualquer momento, com o fim de se manter na liderança tecnológica o Departamento de Defesa Americano criou a *Defense Advanced Reserch Project Agency*, a qual conduziria avançadas pesquisas no ramo militar, o que levou a criação da *ARPANET*, uma rede de computadores de grande escala a qual se comunicava entre si e a mesma não permitia que dados fossem duplicados centralizando eles (A ARPANET, 1997)

Com isto varias outras instituições surgiram com o mesmo propósito de compartilhamento de dados. Para que as diversas redes pudessem se comunicar precisava que fosse criado um protocolo para que não ouve-se divergência na transmissão de dados referentes aos seus protocolos de envio. Com isto foi criado o *Network Control Protocol*, ou NCP, o qual era um conjunto de regras que padronizava a comunicação entre máquinas nas diferentes redes. A partir de este ponto os protocolos foram melhorados até chegar ao que atualmente é utilizado TCP/IP.

Hoje a Internet disponibiliza vários tipos de dados desde textos até vídeos, nos serve tanto como para pesquisas, diversão é trabalho. O método de apresentação disto é por meio de *sites* os quais são desenvolvidos também de diversas formas mais que a parte visual deles é composta de *tag's* HTML e interpretada por um navegador.

Com a evolução da tecnologia da informação, novos dispositivos foram surgindo e cada um com um tipo de resolução de tela diferente o que complicava na hora de acessar as informações dos *sites*, desde uma tela de celular que tem a sua resolução extremamente pequena até televisores os quais oferecem maior resolução, os *sites* antigamente não se

ajustavam as variadas resoluções o que era um grande empecilho para a navegação nos diversos *sites*.

Com o intuito de quebrar estas restrições tanto para o usuário quanto para o desenvolvedor, o qual tinha que refazer o seu trabalho varias vezes para o *site* se adequar as diversas resoluções, surgiu à abordagem da *One Web*, o qual na hora de criar o *site* seja utilizado tecnologias com capacidade de se adaptar as diversas resoluções (CAELUM, 2014).

Para isto são utilizados recursos do CSS3 o qual permite que sejam adequados os layouts de uma única página em varias resoluções. Existem *Frameworks* para facilitar o uso desta tecnologia como o *Bootstrap*.

2.2 JAVAEE

Java é uma linguagem de programação orientada a objetos, lançada ao mercado pela Sun Mycrosistems no ano de 1995. Por conta do seu compilador, o qual gera o seu código independente do sistema da máquina e tem o nome de *bytecode*, Java é multiplataformas e pode ser utilizado em qualquer sistema operacional (ORACLE, 2015).

A portabilidade do Java depende da sua máquina virtual a *Java Virtual Machine* (JVM). Para o que um programa Java consiga ser executado em uma máquina somente e preciso que a JVM esteja instalada na mesma.

Na Figura 1 *MyProgram.java* é um arquivo Java o qual contêm uma sequência de comandos a serem compilados e transformados em *bytecode*. Os *bytecodes* serão armazenados em um outro arquivo que é de extensão .class, e este sim é interpretado pela JVM a qual o transforma a sua vez em binário para a execução do mesmo.

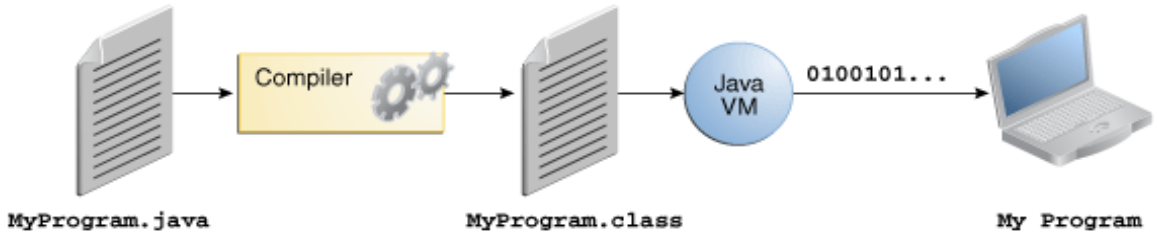

Figura 1 - Transformação de um arquivo Java pelo compilador para ser interpretado pela JVM e executado. Fonte: (JAVANOTES, 2014).

O *bytecode* é um arquivo binário universal e sua principal vantagem é a portabilidade que proporciona ao ser interpretado pela JVM. Já em outras linguagens de programação seria necessária a recompilação do código dependendo de cada sistema operacional. (NETO, 2009).

Para o desenvolvimento de aplicações *Web* e corporativas utilizando a linguagem Java, são necessários recursos inclusos na API básica da linguagem, estes se encontram na versão *Java Enterprise Edition* (JavaEE).

JavaEE consiste em um conjunto de especificações as quais tem o propósito de facilitar o desenvolvimento de aplicações empresariais. Estas especificações solucionam rotinas diárias como, por exemplo, a conexão e as transições com o banco de dados, tratamento de requisições de dados, etc. (CAELUM, 2013).

JavaEE contém variadas APIS, ou especificações, dentre as principais destas se encontram:

- Java Autenthication and Authorization Service;
- Java Naming and Directory Interface;
- Java Message Service;
- Java API for XML *Web* Services, Java API for XML Binding;
- Java Management Extensions;
- Java Server Pages (JSP), Java Servlets, Java Server Faces;
- Enterprise Javabeans Components e Java Persistence API.

Estas APIS, quando implementadas de forma correta no projeto, auxiliam bastante no desenvolvimento, como por exemplo, a *Java Persistence API* que auxilia com a comunicação e transações entre a aplicação e o Banco de Dados escolhido.

2.3 PROTOCOLO HTTP

O JavaEE é geralmente utilizado para desenvolver aplicações as quais utilizam protocolo como HTTP e HTTPS. O protocolo HTTP não retém informações das requisições efetuadas pelo cliente, cada requisição e efetuada de modo independente (TOTTY et al., 2002).

Atualmente as requisições HTTP trabalham com os seguintes métodos: GET, POST, PUT, DELETE, TRACE, CONNECT, HEAD e OPTIONS. Mas não todos os servidores aceitam estes métodos (TOTTY et al., 2002).

Os métodos de requisição utilizados de forma correta oferecem uma maior segurança ao sistema. Cada método tem a sua utilização:

- O método GET utiliza da sua própria URI (comumente chamada de URL) para o envio de dados, ao se enviar uma requisição via GET para o servidor o método cria

a sua especifica URI anexando junto a URL as informações enviadas separando elas por meio de um ponto de interrogação (W3. 2013).

- O método POST envia os dados colocando eles dentro da mensagem, deixando a *URI* limpa sem mostrar os dados, por este motivo e possível enviar qualquer tipo de dados pela requisição sem que fiquem visíveis. Ele é muito utilizado em requisições de acesso (W3. 2013).
- O método PUT é utilizado para atualizar determinados dados alterados mediante um formulário, ao enviar a requisição ele oculta as informação da mesma forma que o método POST faz (W3. 2013).
- O método DELETE envia uma requisição para eliminar o determinado item que o cliente deseje que deixem de existir do sistema.
- O método HEAD retorna informações sobre um recurso, ele é muito similar ao método GET, somente não retorna as informações no corpo da URI.

JavaEE roda em cima de um servidor chamado servidor de container, existem vários é um dos mais famosos é o Apache Tomcat o qual implementa varias especificações do JavaEE, mas é principalmente focado em JSP e Servlets.

No sistema de bancas de TCC é utilizado este servidor por causa da grande facilidade no manejo e também por contemplar toda a necessidade do sistema no que se refere às partes de requisições *Web* e apresentação das páginas JSP.

2.4 JAVA SERVER PAGES JSP

Desenvolvida pela Sun Microsystems, é fundamentada em SSI (*Server Side Includes*), é um tipo de extensão para comandos HTML, o seu conteúdo estático é em HTML e utilizado um servidor de páginas para o seu conteúdo dinâmico (JUNEAU, 2013).

Por serem baseadas em Java, são portáveis podendo funcionar em diferentes sistemas operacionais. É utilizado para o desenvolvimento de aplicações *Web*, atuando como apresentação do sistema.

Trabalha com requisições e respostas, onde o cliente faz a requisição e o servidor envia à resposta, tem como intermediário um navegador de internet. A requisição efetuada pelo cliente e enviada através do navegador para um servidor *Web*, onde a será tratada e reconhecida pelo *JSP Engine* (Zambon, 2012).

Java Server Pages ou JSP é uma linguagem de *script* que tem como objetivo a geração de conteúdo dinâmico para páginas de internet. O JSP pode ser integrado com HTML, já que o HTML é estático no seu conteúdo o JSP terá que efetuar o dinamismo da página (ANSELMO, 2005).

O JSP foi criado pela *SUN* e distribuído de forma aberta, o mesmo precisa um servidor para rodar já que este é *Server-side script*, um usuário de algum sistema que contenha esta tecnologia não consegue ver os códigos JSP já que o mesmo é convertido em HTML pelo servidor (ANSELMO, 2005).

O JSP é formado por HTML e com códigos Java inseridos por meio *tag's* e tem o seguinte funcionamento:

O servidor receve a requisição para uma página JSP, ele interpreta a página e transforma o código em HTML a qual é retornada ao cliente mostrando o resultado da requisição. Isto é efetuado em tempo real, deixando todo o trabalho para o servidor (ANSELMO, 2005).

2.5 APACHE TOMCAT

Este servidor de container criado no ano de 1999 e foi doado pela Sun Microsystems para o Apache Software Foundation e sua primeira versão foi a 3.0, criada com a ajuda de inúmeros especialistas da área e grandes fundações de software. Ele é um servidor open source (Apache Tomcat, 2015).

O servidor de container Apache Tomcat contém uma hierarquia onde uma instância do mesmo é a mais alta no nível hierárquico. Uma instância do Apache Tomcat consiste em um grupo de aplicações de container com nivelações em base da hierarquia.

Alguns dos mais importantes componentes da arquitetura Apache Tomcat:

- *Server*: Este representa todo o *Catalina Servlet Engine* e é mantido como um elemento de mais alto nível dentro de uma instância do Apache Tomcat;
- *Service*: Pode conter um ou mais *Connector* os quais compartilham de uma única *Engine*;
- *Connector*: Este define as classes que efetuam o gerenciamento de requisições e respostas a uma chamada de uma aplicação;
- *Engine*: Este gerencia as requisições recebidas por todos os *Connector;*
- *Host*: O mesmo define os *hosts* virtuais que são contidos dentro de cada instância;

- *Context*: Cada um deste elemento é a representação de uma aplicação *Web* (VUKOTIC; GOODWILL, 2011).

2.6 MYSQL

O MySql, criado pela empresa TcX em 1996, teve o seu inicio com a necessidade desta empresa em ter um banco de dados de alta performance, estável e que funcionasse em *Hardware's* de baixo custo. Na época de sua criação, se tinha o objetivo de que ele funcionasse no sistema operacional Linux, o que fez com que o Sistema Gerenciador de Bancos de Dados (SGBD) popularizasse em 1999 . O mesmo possui licenciamento GNU *General Public License* (SUEHRING, 2002).

O MySql conta com uma segurança, contendo integridade de dados, *backup* e restauração de dados, controle de acesso e de usuários e contém verificações de correção, necessárias na hora em que alguma tabela seja corrompida.

O MySQL pode ser utilizado tanto em aplicações *Web* quanto em aplicações desktop, integrando com as principais linguagens de programação como Java, Delphi, C# entre outras (SUEHRING, 2002).

2.7 VRAPTOR

VRaptor é um *Framework* MVC criado em 2003 no IME-USP, o qual se encontra na sua versão quatro e utiliza o CDI 1.1. Ele é um *Framework* para desenvolvimento ágil em Java, ele utiliza injeção de dependências, modelo REST e convenção sobre configuração, como por exemplo, para criar uma classe controladora somente precisa ser anotada com *@Controller* que o VRaptor usa suas convenções de JSP e URL'S para configurar ele automaticamente (VRAPTOR, 2015).

2.8 MAVEN

Maven é uma ferramenta de gerenciamento de dependências pertencente a Apache auxilia na construção do projeto. Desenvolvido para projetos em Java conta com uma vasta biblioteca de *JARS*. Estas dependências são gerenciadas a partir de um XML chamado POM Figura 2 (What is Maven?, 2015).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
    <modelVersion>4.0.0</modelVersion>

    <groupId>br.edu.utfpr</groupId>
    <artifactId>est-tads-esteban</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>est-tads-esteban</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <weld.version>2.1.2.Final</weld.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.33</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.metro</groupId>
            <artifactId>webservices-rt</artifactId>
            <version>2.3</version>
        </dependency>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>
```

Figura 2 - Arquivo pom.xml onde se encontram as dependências do projeto.
Fonte: Própria.

Neste exemplo do pom.xml se encontram os seguintes items:

- *groupId*: Localiza onde se encontra o package principal*;*
- *artifactId*: Nome do Projeto;
- *version*: Número da versão do projeto;
- *packaging*: Tipo do projeto.

Após estes itens de cabeçalho se encontram os itens:

- *dependencies*: Tag onde inicia a identificação de das dependências;
- *dependency*: Tag para identificação de uma nova dependência;
- *groupid*: Nome da dependência;
- *artifactId*: Nome da dependência dentro do repositório;
- *Version*: Numera da versão da dependência.

Com este arquivo de configuração certo Maven efetuara o trabalho de gerenciamento das dependências automaticamente ao construir o projeto (What is Maven?, 2015).

2.9 REPRESENTATION STATE TRANSFER (REST)

O modelo REST utiliza um conjunto de interfaces genéricas para fazer interações sem estado, por meio da transferência de representações de recursos. As três principais restrições definidas pelo REST se referem diretamente a aplicação para a *Web*:

- Identificação global: todos os recursos são identificados através do mesmo mecanismo global. Cada recurso disponível na *Web* deve ser identificado utilizando uma URL;
- Interfaces uniformes: a interação entre os agentes é feita com recurso ao protocolo HTTP;
- Interações *stateless*: o estado concreto de uma aplicação *Web* é mantido com base no estado de um conjunto de recursos. O servidor conhece o estado dos seus recursos mas não mantém informação sobre as sessões dos clientes (Architectural Styles and the Design of Network-based Software Architectures, 2000).

Geralmente utilizado em *Web Services* o serviço REST permite a conexão ou comunicação entre diferentes *URL* ou diferentes tipos de aplicações, neste se encontram vários tipos de comunicação mais geralmente é utilizada o envio de informações via JSON gerado pela própria aplicação ou manualmente. A comunicação é efetuada via protocolo HTTP e cada transação é independente (Architectural Styles and the Design of Network-based Software Architectures, 2000).

2.10 ANDROID

Android é uma plataforma que surgiu de um grupo de organizações que se dedicaram a criação de um telefone móvel melhor, este grupo é denominado *Open Handset Alliance*, liderado pela empresa *Google* e conta com empresas de telefonia móvel, desenvolvedores de telefones móveis, provedores de plataformas entre outros. A plataforma *Android* é um *software* livre (Introdução ao desenvolvimento *Android*, 2016).

O primeiro telefone móvel com a plataforma *Android* rodando em ele foi o aparelho G1 da empresa HTC e fornecido pela empresa T-Mobile. Conforme a data do lançamento do G1 se aproximava a Google liberou o SDK, versão 1.0 plataforma para desenvolvimento de aplicações para *Android* onde mesmo antes do lançamento começaram a surgir varias

aplicações. Como estímulo, a Google patrocinou duas edições do "*Android Developer Challenges*", onde milhões de dólares foram investidos para o incentivo as aplicações, alguns meses depois foi lançado o G1 junto ao *Android Market* atual *Play Store* (Introdução ao desenvolvimento *Android*, 2016).

3 MATERIAL E MÉTODOS

No decorrer deste capitulo será abordado o desenvolvimento *Web* do projeto, serão apresentadas e descritas as ferramentas utilizadas durante o desenvolvimento do projeto *Web*, para isto serão abordados os assuntos relacionados à lógica do projeto com JavaEE e os seus respectivos *Frameworks* tanto para o desenvolvimento visual quanto lógico.

No desenvolvimento do sistema de bancas de TCC, foi usado do *Framework Bootstrap* para este se adequar aos diversos dispositivos e também para deixar o *site* com uma melhor aparência.

No decorrer do desenvolvimento deste projeto foram utilizadas tecnologias para facilitar o desenvolvimento do sistema de bancas de tcc, estás tecnologias serão apresentadas a seguir. A linguagem de programação utilizada neste projeto foi Java na sua versão sete, pela ampla versatilidade e compatibilidade com diversos sistemas operacionais, conta também com uma vasta biblioteca de *Frameworks* que simplificam o desenvolvimento e ampliam as possibilidades de integração com diversos sistemas. Um destes *Frameworks* é o VRaptor utilizado na sua versão 4 o qual facilita a implementação *Web* integrando os *JSP* com HTML 5 e com Bootstrap, também é um *Framework Model View Controller* (MVC) o que facilita na organização e descrição do que cada classe do sistema ira fazer.

Para o desenvolvimento *Web* foi utilizada a IDE NetBeans na sua versão 8 a qual é uma plataforma de desenvolvimento integrada que permite o desenvolvimento de aplicações em varias linguagens, neste caso em Java, é que conta com uma variada biblioteca de *plugins* que ajudam no desenvolvimento. Junto a ele foi efetuada a utilização do servidor de container Tomcat na sua versão 8, o qual interpreta os arquivos JSP e trata as requisições *Web*. Junto foi integrada a ferramenta de Maven que é um gerenciador de dependências, o qual conta com um arquivo XML que contém as bibliotecas a serem utilizadas no sistema.

Para o desenvolvimento na plataforma *Android* foi utilizada a IDE Eclipse com o *plugin* ADT Bundle que permite o desenvolvimento para aplicações *Android* e a linguagem de programação utilizada é Java.

O banco de dados escolhido foi o MySql na sua versão 5.2, por ser um SGBD relacional de uso gratuito.

3.1 O PROCESSO DO TCC

No inicio do semestre letivo o aluno, o qual se inscreveu na matéria de TCC, deve preparar um documento, chamado pré-projeto, com as características do projeto o qual se propõe estudar e apresentar no final do semestre a uma banca formada por quatro professores incluindo o professor orientador, o qual seguiu o projeto do aluno durante o semestre letivo ou em alguns casos, os quais o estudo do aluno dura mais de um semestre letivo, o professor orienta ele durante todo este tempo.

Durante o desenvolvimento deste documento o aluno deve entregar ao professor orientador diversas vezes este documento, com a finalidade do mesmo ler efetuar as correções necessárias para que desta forma o aluno as efetue. Ao fim do tempo, delimitado pela instituição, o aluno deve entregar este documento para ele poder ser avaliado pelo professor responsável pelas bancas.

Caso este pré-projeto seja aceito o aluno passa a desenvolver o seu estudo descrito no documento.

Durante a evolução do estudo, o aluno deve escrever o seu TCC a fim do mesmo ser avaliado diante uma banca a qual dirá se o aluno está aprovado em está matéria ou não.

Para avaliar a banca os professores que se encontram cadastrados nela deverão preencher um documento (Figura 3 e Figura 4) com as notas dadas ao aluno pelo seu estudo. Após este documento é armazenado em um espaço físico determinado a ele.

FICHA DE AVALIAÇÃO - Banca Avaliadora

A banca avaliadora seguirá o sistema de avaliação especificado abaixo. Conceitos:

- **Insatisfatório (< 7,0)**
- **Satisfatório (7,0 a 8,0)**
- **Ótimo (8,0 a 9,0)**
- **Excelente (9,0 a 10,0)**

Acadêmico:

1.Desempenho técnico-científico

Nº	Itens a serem avaliados	Nota (1,0 a 10,0)
1	Argumentação teórica	
2	Metodologia de ensaio utilizada para a realização do trabalho	
3	Suficiência de dados relevantes para a conclusão do trabalho	
4	Conclusões e análise de resultados consistentes com a pesquisa realizada	
5	Apontamento de novos caminhos para a pesquisa e benefícios	
6	Estrutura do texto	
	Subtotal (ST) 01= (1+2+3+4+5+6) / 6	

2.Desempenho na apresentação

7	Domínio do aluno pesquisador referente ao tema de defesa	
8	Dinâmica de apresentação	
9	Ética de apresentação	
	Subtotal(ST) 02 = (7+8+9) / 3	
TOTAL= [(ST 01)×8 + (ST 02)×2] / 10		

Figura 3 - Ficha de Avaliação: Banca Avaliadora.
Fonte: UTFPR.

FICHA DE AVALIAÇÃO - Professor Orientador

A banca avaliadora seguirá o sistema de avaliação especificado abaixo. Conceitos:

- **Insatisfatório (< 7,0)**
- **Satisfatório (7,0 a 8,0)**
- **Ótimo (8,0 a 9,0)**
- **Excelente (9,0 a 10,0)**

Acadêmico:

1.Desempenho técnico-científico

Nº	Itens a serem avaliados	Nota (1,0 a 10,0)
1	Argumentação teórica	
2	Metodologia de ensaio utilizada para a realização do trabalho	
3	Suficiência de dados relevantes para a conclusão do trabalho	
4	Conclusões e análise de resultados consistentes com a pesquisa realizada	
5	Apontamento de novos caminhos para a pesquisa e benefícios	
6	Estrutura do texto	
	Subtotal (ST) 01= (1+2+3+4+5+6) / 6	

2.Desempenho na apresentação

7	Domínio do aluno pesquisador referente ao tema de defesa	
8	Dinâmica de apresentação	
9	Ética de apresentação	
	Subtotal(ST) 02 = (7+8+9) / 3	
TOTAL= [(ST 01)×8 + (ST 02)×2] / 10		

Registro da Média Geral obtida: Uso exclusivo do Professor Orientador

MÉDIA FINAL= (TOTAL 1+2+3) ÷ 3	**MF= _____**

Figura 4 - Ficha de Avaliação: Professor Orientador.
Fonte: UTFPR.

Estes formulários de avaliação (Figura 3 e Figura 4) devem ser preenchidos pelos respectivos professores que farão parte da banca, são duas fichas de avaliação da qual uma delas é especial para o professor orientador do aluno. As respectivas fichas tem um método de avaliação similar, a única diferença se encontra na ficha preenchida pelo professor orientador o qual deve efetuar o calculo de uma nota chamada Media Final.

As fichas de avaliação se encontram separadas em três tópicos, no primeiro tópico se encontra uma referencia a os valores das notas mostrando dando-se a entender que a nota mínima que um aluno pode receber para ser aprovado é 7,0, no tópico dois Desempenho técnico-cientifico se encontram seis itens de avaliação dos quais se referem a metodologia de apresentação e conhecimentos teóricos e práticos aplicados do aluno para à banca, dentre estes a Argumentação teórica do aluno, a preparação do mesmo para apresentar, se as informações que ele apresenta são suficientes para o entendimento do assunto, entre outros. Já no terceiro e ultimo tópico se encontra o Desempenho do aluno na apresentação são três notas que se referem ao domínio do assunto apresentado pelo aluno, a dinâmica o qual tem para apresentar a mesma e a ética com a qual ele apresenta a mesma. Após o segundo e terceiro tópico está localizado o total que seria uma media que o aluno receve de um total das notas de cada tópico já na ficha de avaliação do professor orientador tem a média final que trata de uma media efetuada a partir dos itens um dois e três mais o item três.

3.2 DESENVOLVIMENTO DO MODELO ENTIDADE RELACIONAMENTO

Ao começar o projeto partindo do ponto do Banco de Dados (BD) foi criado um Modelo Entidade Relacionamento (MER) o qual abarcasse a situação de avaliação TCC dos cursos de TADS e CC (Figura 5).

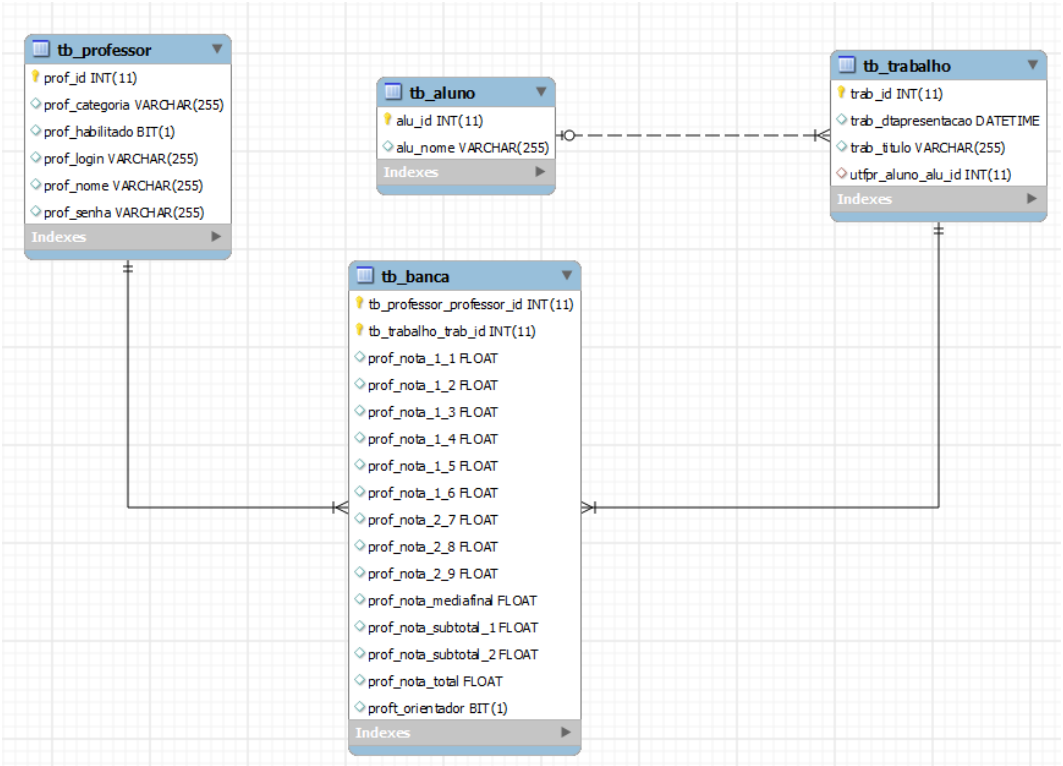

Figura 5 - MER do banco de dados utilizado no sistema.
Fonte: Própria.

As tabelas do BD foram nomeadas da seguinte forma:

- tb_professor: Contém informações relacionadas ao professores cadastrados no sistema;
- tb_aluno: Contém informações referentes aos alunos que serão cadastrados dentro do sistema para apresentar o TCC;
- tb_trabalho: Contém informações referentes ao trabalho a ser apresentado por determinado aluno, dentro desta tabela, a um relacionamento com a tabela aluno para identificação do proprietário do trabalho;
- tb_banca: Contém informações referente a banca que será formada, esta tabela é uma tabela associativa entre tb_professor e tb_trabalho onde se identificara professor, se for orientador ou não, trabalho avaliado, e notas dadas ao trabalho.

A partir do MER foi criado o BD dentro do Sistema Gerenciador de Banco de Dados (SGBD) MySql e colocada dentro da instancia criada no MySql para o sistema poder usar o BD.

3.3 DIAGRAMA DE CASO DE USO

O diagrama de caso de uso apresentado na Figura 6, mostra as funcionalidades do sistema que podem ser executadas pelo administrador/es do sistema. O professor administrador pode cadastrar Alunos, Professores, Trabalhos e Bancas, o mesmo também pode avaliar bancas.

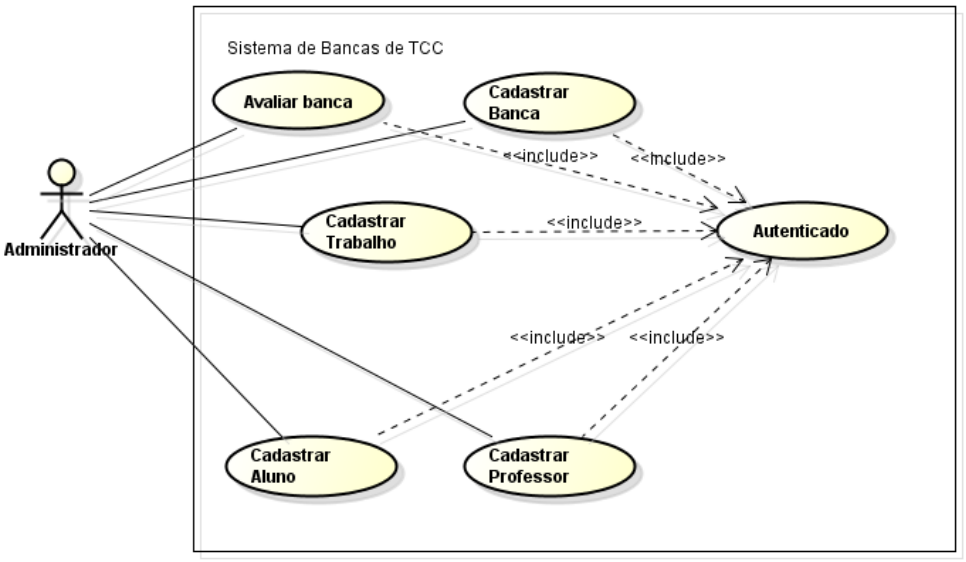

Figura 6 - Diagrama de caso de uso do professor administrador.
Fonte: Própria.

3.4 DIAGRAMA DE CLASSES

O diagrama de classes apresentado na Figura 7, mostra as entidades Aluno, Trabalho, Banca e Professor contidas no sistema, as classes são anotadas com @Entity do *Java Persistence API* (JPA) e todos seus atributos são privados.

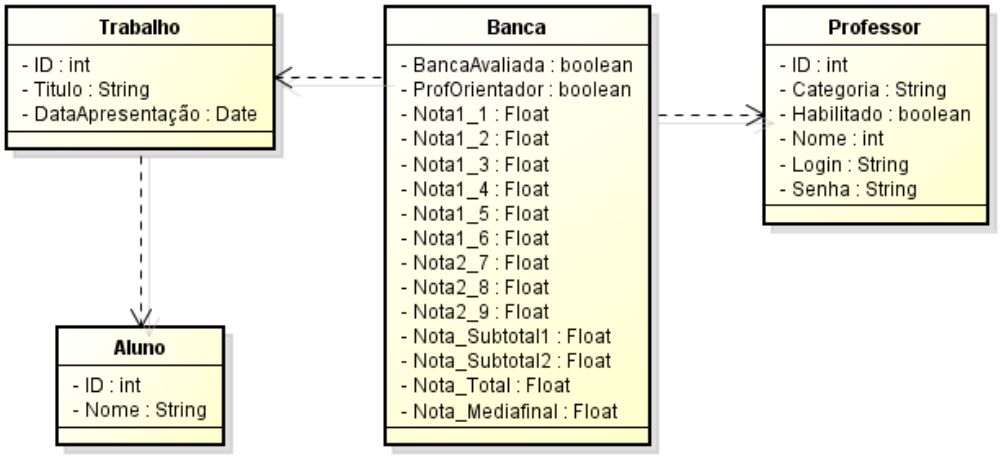

Figura 7 - Diagrama de classes das entidades do sistema.
Fonte: Própria.

3.5 REQUISITOS FUNCIONAIS

São requisitos que compreendem processos fundamentais do sistema desenvolvido apresentados a partir do Quadro 1 até o Quadro 5, estes requisitos compreendem os processos de cadastros de Alunos, Trabalhos, Professores e Bancas, também mostra o processo de Avaliação de Banca.

RF1: Cadastrar Alunos			Oculto ( )	
Descrição: O professor administrador deve cadastrar os alunos que estiverem na material do tcc				
Requisitos Não-Funcionais				
Nome	**Descrição**	**Categoria**	**Desejável**	**Permanente**
RNF1.1 Verificar Cadastro	Deverá ser verificado se o aluno já possui cadastro	Usabilidade	( )	(X)

Quadro 1 - Requisito funcional para cadastro do aluno.

RF2: Cadastrar Trabalhos			Oculto ( )	
Descrição: O professor administrador deve cadastrar os trabalhos que serão apresentados a banca				
Requisitos Não-Funcionais				
Nome	**Descrição**	**Categoria**	**Desejável**	**Permanente**
RNF2.1 Verificar Aluno autor	Deverá ser inserido junto ao cadastro o Aluno autor do trabalho	Usabilidade	( )	(X)

Quadro 2 - Requisito funcional para cadastro do trabalho.

RF3: Cadastrar Banca			Oculto ( )	
Descrição: O professor administrador deve cadastrar os trabalhos que serão apresentados a banca junto aos professores que farão parte da mesma				
Requisitos Não-Funcionais				
Nome	**Descrição**	**Categoria**	**Desejável**	**Permanente**
RNF3.1 Verificar data de apresentação	Deverá ser verificada a data e horario de apresentação do tcc e cadastrar junto	Usabilidade	( )	(X)

Quadro 3 - Requisito funcional para cadastro da banca.

RF4: Cadastrar Professor			Oculto ( )	
Descrição: O professor administrador deve cadastrar os professores que avaliaram bancas e que serão orientadores de trabalhos				
Requisitos Não-Funcionais				
Nome	**Descrição**	**Categoria**	**Desejável**	**Permanente**
RNF4.1 Verificar categoria do professor	Deverá ser verificado se o professor a ser cadastrado será Administrador ou Normal.	Usabilidade	( )	(X)

Quadro 4 - Requisito funcional para o cadastro de Professores.

RF5: Avaliar Banca			Oculto ( )	
Descrição: Os professores deverão efetuar login no sistema para avaliar a banca a qual se encontram cadastrados				
Requisitos Não-Funcionais				
Nome	**Descrição**	**Categoria**	**Desejável**	**Permanente**
RNF5.1 Verificar horario e data da banca	Deverá ser verificada a data e horario de apresentação do tcc e avaliada junto a apresentação da mesma	Usabilidade	( )	(X)

Quadro 5 - Requisito funcional para a avaliação do trabalho.

4 RESULTADOS E DISCUSSÃO

Com a ideia de aperfeiçoar o sistema de avaliação das bancas de TCC foi preciso um estudo de como eram avaliadas as mesmas, por meio deste foi concluído que a papelada era simples mais sujeita a erros humanos já que as notas de médias e notas sub totais eram calculadas pelos professores, mesmo que as contas fossem refeitas antes de dar a média final o tempo consumido pela refeita dos cálculos era alto. Por tanto, foi decido que o próprio sistema efetuaria todos os cálculos, o professor somente deveria inserir as notas de cada categoria e enviar o formulário, que o sistema efetuaria o restante.

4.1 IMPLEMENTAÇÃO DA APLICAÇÃO WEB

Foi utilizado um projeto em branco com VRaptor 4 disponibilizado pelo próprio criador do *Framework* em sua página no GitHub (Figura 8), o mesmo já vem configurado para começar a programação do sistema.

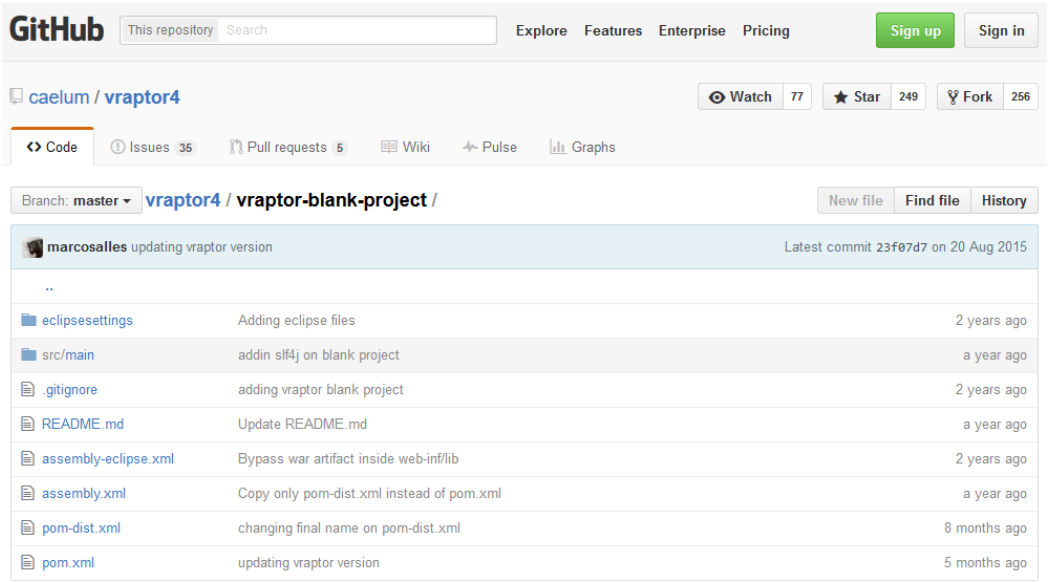

Figura 8 - Projeto em branco configurado com VRaptor 4 disponibilizado no GitHub.
Fonte: GitHub.

Este projeto vem com configurações básicas para o funcionamento do VRaptor 4, as dependências que o Maven ira gerenciar para o inicio já se encontram no pom.xml, o projeto já vem com uma página inicial de Hello Word para o programador testar se as configurações padrão estão de acordo com a sua máquina.

No começo do desenvolvimento foi configurado o arquivo de conexão com o banco de dados MySql chamado persistence.xml (Figura 9), este arquivo é lido na hora de iniciar o sistema para poder reconhecer o BD, a instancia e o nome do mesmo.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/20
    <persistence-unit name="default">

        <properties>
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/est"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.password" value="    "/>
            <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="false" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
        </properties>
    </persistence-unit>
</persistence>
```

Figura 9 - Arquivo persistence.xml de conexão com o banco de dados do sistema.
Fonte: Própria.

Dentro deste arquivo se encontram varias propriedades que definem em si em qual BD será persistido o sistema, as propriedades são:

- javax.persistence.jdbc.url: Link onde o BD se encontra instanciado;
- javax.persistence.jdbc.user: Nome do usuário para acessar o BD;
- javax.persistence.jdbc.Driver: Nome do JAR utilizado para a conexão;
- javax.persistence.jdbc.password: Senha do usuário para acessar o BD.

Com este arquivo configurado é necessário adicionar a dependência do *driver Java Database Connectivity* (JDBC) a se utilizar para a conexão com o BD no arquivo pom.xml.

As classes que acessam o arquivo *persistence.xml* são as classes *Data Access Object* (DAO), que por meio da injeção de dependências, anotada com *@Inject* no seu construtor secundário, efetua a instancia do *EntityManager* para ser gerenciado pela CDI. O *EntityManager* permite a conexão da classe com o BD para assim ser efetuadas as transações com o mesmo, como por exemplo, a classe AlunoDAO (Figura 10).

```java
public class AlunoDAO {

    private final EntityManager manager;

    public AlunoDAO() {
        this(null);
    }

    @Inject
    public AlunoDAO(EntityManager manager) {
        this.manager = manager;
    }

    public void save(Aluno aluno) {
        manager.persist(aluno);
    }

    public Aluno findId(int id) {
        return manager.find(Aluno.class, id);
    }

    public void delete(int id) {
        manager.remove(findId(id));
    }

    public void update(Aluno aluno) {
        manager.merge(aluno);
    }

    public List<Aluno> findAll() {
        Query q = manager.createQuery("SELECT e FROM Aluno e");
        return q.getResultList();
    }
```

Figura 10 - Classe AlunoDAO do sistema utilizada para transações com o banco de dados.
Fonte: Própria.

Como neste exemplo da classe AlunoDAO o *Framework* VRaptor 4 pede que o DAO seja instanciado com um construtor nulo e outro com a anotação da injeção e a instanciação do EntityManager para controle do sistema. Dentro da imagem temos um *Create Remove Update Delete* (CRUD) o qual será utilizado para a persistência dos dados referentes ao Aluno.

O sistema segue o padrão MVC que serve como um separador de pacotes de classes é tem por finalidade separar para especificar melhor as classes, segundo este padrão quando na camada de visão o usuário solicita uma ação a camada de controle é responsável por processar está ação e enviar se utilizar do modelo para poder efetuar a transação no BD necessária em determinados casos ai a camada de visão é responsável pelo envio da mensagem caso a operação tenha sido efetuada com sucesso ou não.

O Modelo é utilizado para referenciar as classes que serão utilizadas para armazenar dados dentro do BD, cada classe dentro deste pacote tem que ser anotadas de forma que o

sistema entenda que se trata de uma classe para armazenamento de dados igual a do BD, como por exemplo, a classe Aluno (Figura 11).

```
@Entity
@Table(name = "tb_aluno")
public class Aluno implements Serializable {

    private static final long serialVersionUID = 1L;

    @TableGenerator(name = "aluno_generator", table = "hibernate_sequences",
            pkColumnName = "sequence_name",
            valueColumnName = "sequence_next_hi_value",
            pkColumnValue = "id_alu", allocationSize = 1, initialValue = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE,
            generator = "aluno_generator")
    @Column(name = "alu_id")
    private int Id;

    @Column(name = "alu_nome")
    private String Nome;

    //bi-directional many-to-one association to TbTrabalho
    @OneToMany(mappedBy = "aluno")
    private List<Trabalho> trabalhos;

    public Aluno() {
    }

    public int getId() {
        return Id;
    }
```

Figura 11 - Classe modelo Aluno do sistema.
Fonte: Própria.

A classe Aluno que se encontra dentro do pacote de Modelo contém varias anotações de configuração para as transações que serão efetuadas em cima dela, dentre elas:

- @Entity: Define que a classe é uma entidade do BD;
- @Table: Utiliza a propriedade *name* para definir o nome da tabela dentro do BD;
- @TableGenerator: Foi utilizado para a geração automática do Id.

Esta classe é utilizada não somente pela classe DAO da mesma, mais por varias outras classes que precisem instanciar um Aluno para trabalhar com ele.

O Controlador da classe Aluno (Figura 12) citada como exemplo, é o responsável por processar as ações que são enviadas pelo usuário a partir da *View*, por exemplo, quando o usuário aciona a página de Alunos automaticamente é efetuado uma busca de todos os Alunos no banco de dados é o resultado enviado a classe AlunoController que se responsabiliza por efetuar o processamento dos dados e os envia para a *View* em forma de texto.

```java
@Controller
public class AlunosController {

    private final Result result;
    private final AlunoDAO dao;

    public AlunosController() {
        this(null, null);
    }

    @Inject
    public AlunosController(Result result, AlunoDAO dao) {
        this.result = result;
        this.dao = dao;
    }


    @Get
    @Path("/alunos")
    public void aluno() {
        result.include("alunos", dao.findAll());
    }


    @Post
    @Path("/alunos/add")
    public void add(Aluno aluno) {
        dao.save(aluno);
        result.forwardTo(this).aluno();
    }
```

Figura 12 - Classe AlunosController do sistema.
Fonte: Própria.

Na Figura 12 podemos observar o *Controller* da classe Aluno, como citado acima por meio da chamada da página Aluno é automaticamente efetuada a busca de todos eles e enviada a *View* por meio da função aluno. Pode-se observar também que temos a função add que é responsável por enviar um aluno novo para o banco de dados por meio da classe AlunoDAO. Também foram efetuados dois controladores um nulo efetuando a chamada do outro controlador que é injetado para o CDI e inicializando as classes *Result* responsável pelo envio de mensagens e resultados para a *View* e a inicialização da classe AlunoDAO como já mencionada responsável pelas transações entre a classe Aluno e o BD.

A *View* da classe Aluno (Figura 13) é responsável por mostrar para o usuário o conteúdo que é processado pela classe AlunoController, a página de *View* foi criada em JSP é utiliza em conjunto *Hipertext Markup Language* (HTML) 5, *Cascading Style Sheet* (CSS) 3 e Bootstrap.

```
<%@ include file="../../../header.jsp" %>
      <div class="container">
       <div class="page-header">
           <h1>Cadastro de Alunos</h1>
       </div>
      <form class="form-horizontal" role="form" action="/est-tads-esteban/alunos/add" method="Post">
       <div class="form-group">
         <label class="control-label col-sm-2" for="email">Nome</label>
         <div class="col-sm-10">
           <input type="text" name="aluno.nome" required="true" class="form-control" placeholder="Insira o nome do Aluno" >
         </div>
       </div>
       <div class="form-group">
         <div class="col-sm-offset-2 col-sm-10">
           <button type="submit" value="submit" class="btn btn-default">Submit</button>
         </div>
       </div>
      </form>

       <hr>
       <h1>Lista de Alunos cadastrados</h1>
       <hr>
       <form class="form-horizontal" role="form">
           <ul class="list-group">
               <c:forEach items="${alunos}" var="a" >
                   <li class="list-group-item"> ${a.nome} </li>
               </c:forEach>
           </ul>
       </form>
       <hr>
      </div>
<%@ include file="../../../footer.jsp" %>
```

Figura 13 - Página Aluno.jsp utilizada para a criação da tela do sistema.
Fonte: Própria.

Esta página criada com JSP contém um formulário para a inserção do aluno no sistema este formulário envia os dados inseridos nele via requisição REST e método POST, após o formulário se encontra uma lista com os alunos já cadastrados no sistema, as informações da lista são enviadas pela classe AlunoController.java e o atributo que se responsabiliza pelo envio é o Result que é inicializado no construtor da classe.

4.2 SISTEMA DE AUTENTICAÇÃO

No sistema foi utilizado um método de autenticação ou *login* de usuário por meio da criação de uma classe que intercepta as URL e verifica se em primeiro lugar o usuário se encontra autenticado no sistema e em segundo se o usuário tem permissão de acesso a aquela página.

Para isto foi criado um interceptador de URL com a anotação do VRaptor 4 *@intercepts* (Figura 14) que funciona da mesma forma que um *Servlet Filter* ele pode interceptar requisições antes que a lógica seja efetuado sobre elas, ou mesmo impedir a ação do *Controller* na requisição sendo redirecionada a outra página.

```java
@Intercepts
public class AuthorizationInterceptor {

    private final UserInfo info;
    private final ProfessorDAO dao;
    private final Result result;
    private final ResourceBundle bundle;

    @Inject
    public AuthorizationInterceptor(UserInfo info, ProfessorDAO dao,
            Result result, ResourceBundle bundle) {
        this.info = info;
        this.dao = dao;
        this.result = result;
        this.bundle = bundle;
    }
    /**
     * @deprecated CDI eyes only
     */
    public AuthorizationInterceptor() {
        this(null, null, null, null);
    }

    @Accepts
    public boolean accepts(ControllerMethod method) {
        return !method.containsAnnotation(Public.class);
    }
```

Figura 14 - Classe AutorizationInterceptor utilizada para interceptar URLS.
Fonte: Própria.

A classe AuthorizationInterceptor tem a funcionalidade mencionada acima, a classe inicia com a anotação *@intercepts* que outorga a classe a funcionalidade de interceptar URL'S, nas variáveis utiliza a classe ProfessorDAO que faz as consultas com o BD. A classe contém a função accepts que faz a verificação do método que esta tentando ser acessado pela requisição contém a anotação *@Public*, esta anotação foi criada com a intenção de manter a página de autenticação aberta a todo publico sem a necessidade de estar autenticado no sistema. A função accepts anotada com *@accepts* serve para controlar quando uma função será interceptada, neste caso sempre que uma função não estiver anotada com *@Public* ela será interceptada, é será enviada para a função intercepts (Figura 15).

```
@AroundCall
public void intercept(SimpleInterceptorStack stack, ControllerMethod method) {

    Professor current = info.getProfessor();
    /*
    try {
        dao.refresh(current);
    } catch (Exception e) {
        // o que acontece se o usuário não existir ou não estiver logado
    }*/

    if (current == null) {
        SimpleMessage msg = new SimpleMessage("Alerta", "Professor não logado");
        msg.setBundle(bundle);
        result.include("errors", asList(msg));
        result.redirectTo(IndexController.class).index();
        return;
    }

    if(!method.containsAnnotation(AllUsers.class)){
        if(current.getCategoria().equals("Normal")){
            SimpleMessage msg = new SimpleMessage("Alerta", "Professor não "
                    + "autorizado contatar o seu Administrador");
            msg.setBundle(bundle);
            result.include("errors", asList(msg));
            result.redirectTo(HomeController.class).home();
            return;
    }
    }

    stack.next();
}
```

Figura 15 - Função intercepts dentro da classe AutorizationInterceptor.
Fonte: Própria.

Esta função e ativada caso uma função requisitada não esteja anotada com a anotação *@Public* esta função anotada *@AroundCall* efetua a verificação no momento que é enviada a requisição, primeiramente ela pega as informações contidas na classe UserInfo e envia para uma nova instancia da classe Professor, caso estas informações sejam nulas será interceptada a requisição e anulada efetuando um redirecionamento para a página Index informando que o Professor não se encontra autenticado. Caso a requisição seja enviada para um método o qual não contém a anotação *@AllUsers* será verificada a categoria na qual o Professor foi cadastrado, caso seja de categoria Normal e não Administrador o usuário será redirecionado para a página Home do sistema e aparecera a mensagem de que o mesmo não tem autoridade para acessar aquela página pretendida e que caso o mesmo tenha interesse de acesso peça ao Administrador do sistema para que seja modificada sua categoria.

A classe UserInfo (Figura 16), mantém os dados do professor durante a sessão para serem consultados quando seja necessário.

```
@SessionScoped
@Named
public class UserInfo implements Serializable {

    private static final long serialVersionUID = 1L;
    private Professor professor;

    public Professor getProfessor() {
        return this.professor;
    }

    public void login(Professor professor) {
        this.professor = professor;
    }

    public void logout() {
        this.professor = null;
    }

    public boolean logado(){
        return this.professor != null;
    }
}
```

Figura 16 - Classe UserInfo utilizada para manter os dados do professor na sessão.
Fonte: Própria.

Está classe é anotada inicialmente com *@SessionScoped* que faz com que os dados da mesma sejam mantidos durante a sessão, após a anotação *@Named* permite que os componentes da classe sejam enviados para *View* para utilização dos mesmos. A classe contém uma variável Professor a qual é *setada* por meio da função *login*, a função *logout* tem por funcionalidade deslogar o usuário do sistema quando o mesmo desejar e a função logado funciona para verificação se está logado ou não.

4.3 AVALIAÇÃO DAS BANCAS

A avaliação das bancas é composta por um formulário *online* (Figura 17) a ser preenchido por todos os integrantes que estão cadastrados em determinada banca, no total são três fichas preenchidas: duas por parte dos professores convidados e uma por parte do professor orientador do aluno.

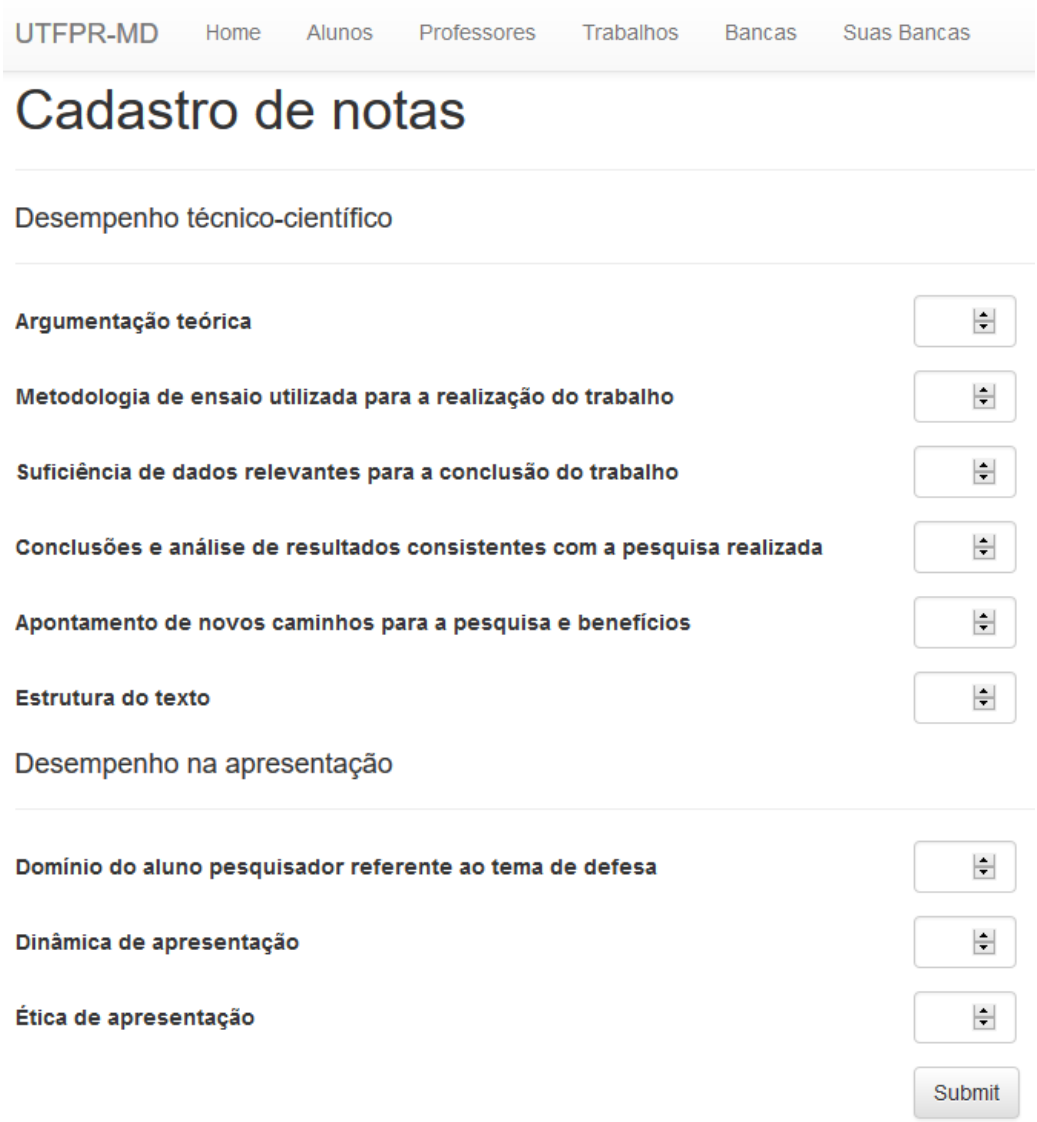

Figura 17 - Tela da Ficha de Avaliação para avaliação do trabalho.
Fonte: Própria.

A ficha de avaliação *online* segue os mesmos padrões da ficha de avaliação apresentadas nas Figuras 3 e 4. As notas devem ser colocadas de 1,0 no seu menor conceito a 10,0 no seu maior conceito.

A ficha do professor orientador pede que o mesmo de uma média final para o trabalho do aluno, sendo calculada pelas soma das duas notas totais dos professores convidados mais a nota total do professor orientador e divido por três. Já no caso da ficha de avaliação *online* não permite isto que o usuário gere qualquer nota total ou subtotal, as notas totais e sub totais são geradas automaticamente pelo sistema assim que o professor efetuar o envio do formulário preenchido já no caso da media final é calculada pelo próprio sistema quando um dos integrantes da banca solicitar que esta seja gerada, mais somente poderá ser efetuada esta solicitação caso todos os integrantes da banca tenham submetido seu formulário.

4.4 DESENVOLVIMENTO ANDROID

Para o desenvolvimento do sistema para a plataforma *Android* foi utilizada a *Integrated Development Environment* (IDE) Eclipse Juno EE com o plugin ADT para desenvolvimento na plataforma *Android*.

O sistema para *Android* foi planejado somente para o cadastramento de notas da banca pelo aplicativo, sendo mais pratico caso o professor avaliador goste. O sistema conta inicialmente com uma página de autenticação do usuário (Figura 18).

Figura 18 - Tela de Autenticação para Android.
Fonte: Própria.

A página de autenticação conta com uma mensagem de boas vindas e dois campos um de Usuário onde será inserido o nome de usuário para acessar o sistema e um campo de senha onde será inserida a senha de acesso ao sistema. Quando o usuário clica no botão *Login* o sistema envia via método REST para o sistema *online* uma requisição de *login* é o sistema vai responder com o código 200 caso os dados estejam corretos e com qualquer outro código caso os dados estejam incorretos.

Quando efetuado *login* corretamente no sistema é armazenado o *httpclient* em uma classe estática para o mesmo se manter até o usuário sair do aplicativo. Ao mesmo tempo é efetuada uma consulta via REST ao sistema *online* requisitando as bancas que o professor tem por avaliar, estás são mostradas em lista (Figura 19).

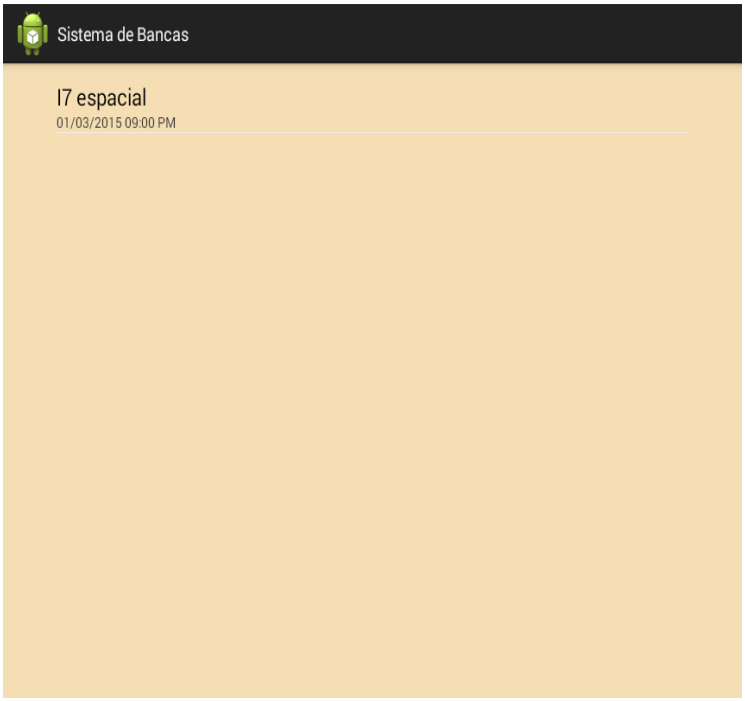

Figura 19 - Tela com Lista de Bancas para Android.
Fonte: Própria.

Na lista são mostrados alguns dados referentes a banca a ser avaliada nome, data de avaliação e horário marcado para a banca. O usuário deverá escolher entre estas bancas para avaliar e será enviado para um formulário parecido com o do sistema *online* onde o mesmo deverá efetuar o cadastro das notas e submeter este formulário que será enviado via REST para o sistema *online* onde será armazenado no BD.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

O VRaptor facilitou o desenvolvimento do sistema, com a utilização da injeção de dependências para o envio de informações para a tela e para as comunicações com o banco de dados, o método de autorização do VRaptor foi eficaz na hora de definir os acessos a páginas por diferentes tipos de categorias de professores.

O sistema trata todas as suas funções via REST o que facilitou muito o desenvolvimento para *Android*, já que somente deveria colocar algumas anotações em determinadas funções para que o mesmo conseguisse se receber e enviar dados. O mais interessante desta etapa é que o sistema está pronto para receber e enviar dados para outras plataformas via REST, poderia ser efetuada uma versão desktop e também desenvolver o sistema para IOS ou Windows Mobile sem precisar efetuar modificações no sistema.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

O sistema conta com uma estrutura REST vasta o que resultaria muito facilmente na migração do sistema para IOS, Windows Phone e Desktop. Criar uma estrutura de relatórios mais vasta seria muito interessante já que atualmente somente se tem um relatório de nota final.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ALEKSA VUKOTIC, JAMES GOODWILL. **Apache Tomcat 7**. Apress, 2011

APACHE MAVEN PROJECT. What is Maven. Disponível em:<https://maven.apache.org/what-is-maven.html>. Acesso em 12 dec. 2015.

APACHE TOMCAT. Apache Tomcat. Disponivel em:<http://tomcat.apache.org/> Acesso em: 10 out. 2015

CAELUM. Apostila Desenvolvimento Web com HTML, CSS e JavaScript. Disponivel em:<http://www.caelum.com.br/apostila-html-css-javascript/web-para-dispositivos-moveis/#7-1-site-mobile-ou-mesmo-site> Acesso em: 07 out. 2015.

CAELUM. Apostila Java para Desenvolvimento Web. Disponivel em: <http://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/#3-1-como-o-java-ee-pode-te-ajudar-a-enfrentar-problemas> Acesso em: 07 out. 2015.

FERNANDO ANSELMO. **Tudo sobre a JSP – com o Netbeans em Aplicações Distribuídas**: Visual Books, 2005.

IBM. Introdução ao desenvolvimento android. Disponível em <https://www.ibm.com/developerworks/br/library/os-android-devel/ > Acesso em 02 mar. 2016

MYSQL. About MySql. Disponivel em:<https://www.mysql.com/about/> Acesso em 09 out. 2015

NETMARKSHARE. Disponivel em: <http://www.netmarketshare.com/> Acesso em: 13 Jun. 2016.

NETO, O. M. **Entendendo e Dominando o Java**. [S.l.]: Digerati Books, 2009.

ORACLE. JavaEE at a Glance. Disponivel em: <http://www.oracle.com/technetwork/java/javaee/overview/index.html> Acesso em 08 out. 2015.

ORACLE. JAVA. Disponível em: <http://www.oracle.com/br/technologies/java/overview/index.html>. Acesso em 09 dec. 2014.

ORACLE. THE HISTORY OF JAVA TECHNOLOGY. Disponível em: <http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html> . Acesso em 09 dec. 2014.

ROY THOMAS FIELDING. Architectural Styles and the Design of Network-based Software Architectures. Disponivel em:

<http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm> Acesso em: 15 dec. 2015.

STEVE SUEHRING. **MySql a Bíblia**. ELSEVIER EDITORA, 2002.

TOTTY B. et al. **HTTP - The Definitive Guide**. 1. ed. [S.l.]: O'Reilly Media, Incorporated, 2002

USP. A ARPANET. Disponivel em: < https://www.ime.usp.br/~is/abc/abc/node20.html> Acesso em 06 out. 2015