Gostou da Aula? - Desenvolvimento de uma aplicação para dispositivos móveis que permite ao aluno Unifieo avaliar as aulas ministradas por seus professores diariamente

Alex Felipe Vieira

Monografia de Iniciação Científica apresentada
ao
Centro Universitário Unifieo

Curso: Ciência da Computação

Orientador: Prof. Me. Rodrigo Bossini Tavares Moreira

Osasco, junho de 2016

Gostou da Aula? - Desenvolvimento de uma aplicação para dispositivos móveis que permite ao aluno Unifieo avaliar as aulas ministradas por seus professores diariamente

Esta versão da monografia contém as correções e alterações sugeridas pelos professores avaliadores após a apresentação do trabalho, realizada em 15/06/2016. Uma cópia da versão original está disponível no Núcleo de Comissões do Centro Universitário UNIFIEO.

Professores Avaliadores:

- Prof. Me. Rodrigo Bossini Tavares Moreira (orientador) - UNIFIEO
- Prof. Dr. Nome Completo - Instituição
- Prof. Dr. Nome Completo - Instituição

Agradecimentos

Você pode fazer agradecimentos aqui, se quiser. Se não quiser, basta comentar a página.

Resumo

VIEIRA, A. F., BOSSINI, R. **Gostou da Aula? - Desenvolvimento de uma aplicação para dispositivos móveis que permite ao aluno Unifieo avaliar as aulas ministradas por seus professores diariamente**. Monografia de Iniciação Científica - Centro Universitário UNIFIEO, Osasco, 2016.

A qualidade do processo de ensino e aprendizagem depende, entre outras coisas, de interação constante entre mestre e aprendiz. Alguns detalhes podem fazer com que, muitas vezes, essa interação não ocorra. Após assistir a uma aula, a avaliação de um aluno sobre ela pode ser boa ou ruim. Embora seja importante que o aluno expresse essa opinião, a fim de que o professor possa melhorar os pontos em que possivelmente falhou, ele pode simplesmente deixar de fazê-lo por diferentes razões. Por exemplo, o aluno pode sentir-se desconfortável caso sinta necessidade de falar sobre pontos negativos que identificou. Independente de ser positiva ou negativa, pode ser que o aluno tenha uma observação muito relevante mas deixe de emiti-la no momento da aula para não a interromper, e depois ela acabe caindo no esquecimento. O professor, por sua vez, ao ministrar uma aula, embora o conteúdo seja essencialmente o mesmo, pode personalizar a forma da oferta dependendo do perfil da turma, o que evidentemente depende de ele conhecê-la, o que pode acontecer mais facilmente caso o aluno tenha meios para emitir suas opiniões sem necessariamente ser identificado. O aplicativo *Gostou da Aula?* visa oferecer ao aluno um meio para avaliação das aulas que acabou de assistir. Trata-se de um aplicativo para dispositivos móveis com Android que oferece um questionário pré-definido que o aluno usará para avaliar diferentes aspectos das aulas. O professor, por sua vez, poderá visualizar as avaliações realizadas, obtendo uma visão geral sobre aspectos positivos e negativos para cada aula que ministra. Evidentemente, as avaliações serão sempre anônimas e o professor não poderá associar qualquer avaliação ao aluno que a tenha realizado.

Palavras-chave: Avaliação de ensino, Dispositivos Móveis, Ensino/Aprendizagem.

Abstract

VIEIRA, A. F., BOSSINI, R. **Gostou da Aula? - Development of a mobile application that allows Unifieo students to evaluate the classes given by their teachers daily.** Undergraduate research thesis - Centro Universitário UNIFIEO, Osasco, 2016.

Teaching and learning process quality depends, among other things, on constant interaction between teacher and student. Often, some details may cause that interaction not to occur. After attending a class, the opinion of a student about it may be good or bad. Although it is important that the student expresses it so the teacher may improve some aspects, s/he may simply not do it for different reasons. For example, the student may feel unconfortable in case his/her opinion includes negative points about the class. Either being positive or negative, it may be that the student has an opinion that is extremely relevant but does not emit it during the class so as not to interrupt it and after that it is simply forgotten. The teacher, in turn, while in a class, although its content is mostly the same, may customize the way the class is taught depending on the students profile, which evidently depending on how well s/he knows it, which may happen more easily if the students have means to emit their opinions without necessarily being identified. The *Gostou da Aula?* application aims on offering students means for evaluating the classes just attended. It is a mobile app for Android devices that offers a predefined questionnaire that students will use to evaluate different aspects of the class. The teacher will be able to visualize the evaluations done so far, obtaining an overview about positive and negative aspects for each class under his or her responsibility. Of course, evaluations will always be anonymous and teachers will not be able to associate any evaluation to the student who made it.

Keywords: Education Evaluation, Mobile Devices, Teaching/Learning.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A qualidade do processo de ensino e aprendizagem depende, entre outras coisas, de interação constante entre mestre e aprendiz. Alguns detalhes podem fazer com que, muitas vezes, essa interação não ocorra. Por exemplo, após assistir a uma aula, a avaliação de um aluno sobre ela pode ser boa ou ruim. Embora seja importante que o aluno expresse essa opinião, a fim de que o professor possa melhorar os pontos em que possivelmente falhou, ele pode simplesmente deixar de fazê-lo por diferentes razões. Além disso, o aluno pode sentir-se desconfortável caso sinta necessidade de falar sobre pontos negativos que identificou. Pode ser também que ele tenha uma opinião muito relevante mas deixe de emiti-la no momento da aula para não a interromper, e depois ela acabe caindo no esquecimento. O professor, por sua vez, ao ministrar uma aula, embora o conteúdo seja essencialmente o mesmo, pode personalizar a forma da oferta dependendo do perfil da turma, o que evidentemente depende de ele conhecê-la, o que pode acontecer mais facilmente caso o aluno tenha meios para emitir suas opiniões sem necessariamente ser identificado. O aplicativo *Gostou da Aula?* visa oferecer ao aluno avaliar as aulas que acabou de assistir. Trata-se de um aplicativo para dispositivos móveis com Android que oferece um questionário pré-definido que o aluno usará para avaliar diferentes aspectos das aulas. O professor, por sua vez, poderá visualizar as avaliações realizadas, obtendo uma visão geral sobre aspectos positivos e negativos para cada aula que ministra. Evidentemente, as avaliações serão sempre anônimas e o professor não poderá associar qualquer avaliação ao aluno que a tenha realizado.

1.1 Justificativa

O desenvolvimento deste projeto implica em uso de tecnologias atuais no mercado, como aquelas relacionadas a aplicações para dispositivos móveis e computação na nuvem, essenciais aos egressos do curso de Ciência da Computação. Além disso, a documentação servirá como material didático a ser consultado por futuros alunos também interessados na tecnologia utilizada. Além disso, o projeto pode ter aplicabilidade real dentro do Centro Universitário FIEO e, a médio e longo prazo, auxiliar a geração de relatórios que evidenciam a situação atual da relação entre a instituição e seus alunos.

1.2 Objetivos

Nesta seção destacamos o objetivo geral do trabalho, o qual a seguir é descrito mais detalhadamente, decomposto em objetivos específicos.

1.2.1 Objetivo Geral

O objetivo geral é desenvolver um aplicativo para dispositivos móveis com Android que permite ao aluno Unifieo avaliar a aula que acabou de assistir.

1.2.2 Objetivos Específicos

- Desenvolver uma aplicação para Android com dois perfis diferentes. O perfil aluno permite ao aluno avaliar a aula que acabou de assistir. O perfil professor permite ao professor verificar as avaliações que suas aulas receberam.
- Desenvolver uma aplicação hospedada em um serviço de computação na nuvem, a qual será responsável pelo armazenamento dos questionários e dados de avaliações realizadas utilizando o aplicativo móvel.

1.3 Contribuições

As principais contribuições deste trabalho são as seguintes:

- Disponibilização de uma aplicação para dispositivos móveis com Android que permite a avaliação de aulas assistidas por alunos do Unifieo.
- Levantamento de indicadores que revelam aspectos fundamentais para a elaboração de questionários úteis e de acordo com princípios básicos de ética.

1.4 Organização do Trabalho

No Capítulo 2, apresentamos os principais conceitos envolvidos na elaboração deste projeto, bem como os componentes tecnológicos utilizados. No Capítulo **??** descrevemos detalhadamente o desenvolvimento técnico do projeto. A descrição fornecida permite ao leitor interessado reproduzir inteiramente o desenvolvimento realizado. No Capítulo 4 o autor indica as principais disciplinas que cursou em Ciência da Computação no Unifieo e discute sobre a importância de cada uma para a elaboração desse trabalho. No Capítulo 5 apresentamos nossas principais conclusões após o término do desenvolvimento do projeto e no Capítulo 6 apresentamos sugestões de trabalhos que podem ser elaborados a partir deste.

Capítulo 2

Conceitos e Tecnologia

Neste capítulo abordamos os principais conceitos envolvidos durante o desenvolvimento do projeto proposto. Além disso, um conjunto amplo de tecnologias utilizadas atualmente no mercado foi empregado, o qual também é descrito.

2.1 Conceitos

Esta seção é dedicada a descrições detalhadas independentes de implementação ou tecnologia sobre os principais conceitos envolvidos durante a elaboração do projeto.

2.1.1 Modelagem de Dados Conceitual

O desenvolvimento de um sistema computacional que utiliza armazenamento de dados em meio persistente pode ocorrer de diversas formas. Uma possível abordagem, muito utilizada no mercado atualmente, consiste na elaboração de uma descrição sobre os dados com os quais a aplicação irá trabalhar. Essa descrição pode ser textual ou gráfica e ela é elaborada de forma independente de qualquer tecnologia. Em geral, ela é conhecida como modelo conceitual. A modelagem conceitual realizada neste trabalho é descrita no capítulo 2.2.2.

2.1.2 Modelagem de Dados Relacional

Uma vez obtida a descrição conceitual dos dados, a sua implementação pode ser realizada de diferentes formas. Entre as mais utilizadas no mercado atual está a modelagem relacional, que consiste no uso de tabelas para o armazemamento dos dados, além de diferentes restrições, como chaves primárias e estrangeiras.

Criar subseções para os seguintes itens: linguagem SQL e álgebra relacional (é só para descrever o que é, sem mostrar código) orientação a objetos Mapeamento Objeto Relacional Web Services Dispositivos Móveis e aplicações

2.1.3 SQL e álgebra relacional

Structured Query Language (linguagem de consulta estruturada) é uma linguagem de pesquisa padrão para banco de dados relacionais, onde permite a comunicação com um banco de dados para a realização de consultas simples ou com álgebra relacional. A álgebra relacional é uma teoria de como podemos manipular as consultas realizadas utilizando o SQL, como por exemplo, mencionar quais são os atributos e filtros que necessários com o intuito de devolver o resultado esperado.

$$\sigma_{id=1}(Empregado) \tag{2.1}$$

2.1.4 Orientação a objetos

O paradigma de POO (programação orientada a objetos) é um modelo para o desenvolvimento de software ao qual utilizamos unidades chamadas de objetos. Esse paradigma nos permite modelar um sistema computacional por meio desses objetos que representam todas as entidades contidas no sistema que, por sua vez, refletem as abstrações de um objeto da vida real para a aplicação, como por exemplo, na modelagem para essa aplicação, foi necessário modelar a entidade aula que representa uma aula da vida real que compõe um aluno, um professor e qualquer outra informação que seja necessária para essa entidade.

Os objetos dessas entidades são identificadas como classes que possuem estados (atributos) e comportamentos (métodos) que definem suas responsabilidades e razão de existir.

```
1 class Aula {
2
3   Aluno aluno;
4   Professor professor;
5
6   //atributos e métodos
7
8 }
9
10 class Professor {
11
12   List<Aula> aulas;
13
14   //atributos e métodos
15
16 }
17
18 class Aluno {
19
20   List<Aula> aulas;
21
22   //atributos e métodos
23
24 }
```

Listing 2.1: *Exemplo de como podemos criar 3 classes e adicionar seus atributos*

Além disso, na POO, temos grandes benefícios devido aos recursos de encapsulamento, herança, polimorfismo que permitem o desenvolvimento de um sistema mais coerente, seguro e manutenível.

2.1.4.1 Encapsulamento

O encapsulamento é um recurso que tem o objeto de esconder todos os passos que são realizados para apenas uma chamada de método. Podemos verificar um pequeno exemplo de encapsulamento durante uma chamada

```
1 class CalculadorDeBonus {
2
3   double calcularBonus(double salario){
4     if(salario < 1000){
5       return salario * 0.15;
6     } else {
7       return salario * 0.10;
8     }
9   }
10
11   //atributos e métodos
12
13 }
```

```
14
15 class Funcionario {
16
17   double salario;
18   double bonus;
19
20   double getBonus(){
21     bonus = new CalculadorDeBonus().calcularBonus(this.salario);
22     return bonus;
23   }
24
25 }
```

Listing 2.2: *Exemplo de utilização do encapsulamento para calcular um bonus*

Observe que a classe Funcionario possui os atributos salario e bonus e o método getBonus(). Quando alguém, que não seja a classe Funcionario chamar o método getBonus(), saberá como esse bonus é calculado? A resposta é não! Pois todos os passos, procedimentos ou qualquer que seja a rotina realizada para calcular o bonus está escondida! Além disso, observe que a classe Funcionario faz uma chamada ao método calcularBonus(**double** salario) que também possui seus procedimentos, ou seja, passos que nem mesmo a classe Funcionario sabe, ou seja, outro exemplo de encapsulamento! É fácil compreender que o encapsulamento é justamente esconder o código para que todos que o chamem, não precisem saber como funciona por de trás dos panos, porém, quais são os benefícios que temos ao utilizá-lá?

- Segurança.
  - Não exibir como tudo é feito por de trás dos panos.
  - Não permitir que alterem qualquer valor que esteja dentro do método.
- Manutenabilidade.
  - Se um dia os procedimentos, passos ou qualquer parte do código encapsulada precisarem ser alterados(as), basta apenas alterar dentro de um único método.
- Reutilização de código.
  - Permite a possibilidade e reutilizar o mesmo código, exigindo apenas uma chama à função que encapsulou todo algoritmo.

2.1.4.2 Herança

A herança é um dos principais recursos da orientação a objetos, pois, por meio dela, podemos criar classes filhas de uma classe. Quando dizemos que uma classe é uma filha de outra, significa que a classe filha herda todos os atributos e métodos da classe mãe, ou seja, ela possuirá o mesmo comportamento que sua classe mãe, porém, ela também terá os seus próprios comportamentos. Por exemplo, na aplicação, fizemos uma abstração de um usuário qualquer no sistema, o resultado obtido dessa abstração foi a classe Usuario:

```
1 public class Usuario {
2
3   long id;
4   String senha;
5   String nome;
6   String sobrenome;
7
8   //métodos e atributos
9
10 }
```

Listing 2.3: *Exemplo de uma classe que representa um usuário no sistema*

Essa classe, como podemos ver, pode representar qualquer usuário dentro da aplicação, porém, nessa aplicação, existem 2 tipos de usuários, que são os alunos e professores, que consequentemente, possuem comportamentos diferentes no sistema, mas, além de suas diferenças, eles possuem semelhanças que são os atributos id, senha, nome, sobrenome ou qualquer outro tipo de comportamento ou estado que é esperado de qualquer usuário.

Temos diversas formas de resolver essa distinção dentro de um sistema computacional, porém, uma das soluções possíveis e sofisticada dentro da orientação a objetos, é fazer com que a classe Professor e Aluno herdem da classe Usuario, pois, dessa forma, faremos com que ambas possuam os mesmo comportamentos de um usuário qualquer, mas, que cada uma tenha o seu próprio comportamento dentro da aplicação. Para fazer com que uma classe herde de outra, basta utilizar a instrução **extends**:

```
1 public class Usuario {
2
3   long id;
4   String senha;
5   String nome;
6   String sobrenome;
7
8   //métodos e atributos
9
10 }
11
12 public class Aluno extends {
13
14   //atributos, métodos
15
16 }
17
18 public class Professor extends {
19
20   //atributos, métodos
21
22 }
```

Listing 2.4: *Exemplo de como criamos classes filhas utilizando a herança do Java*

A partir do momento em que as classes Aluno e Professor estendem da classe Usuario, significa que, além de representarem elas mesmos dentro da aplicação, elas também são um usuário, ou seja, a partir desse momento, qualquer comportamento e estado que estiver na classe Usuario, serão herdados por essas classes.

2.1.4.3 Polimorfismo

Um dos principais pontos da herança em orientação a objetos, é a questão em que, uma classe filha reutilize um comportamento herdado, porém, que precise de alguma moficação peculiar daquela classe, por exemplo, imaginemos que a classe Usuario contenha o método cadastrarSenha(String senha), esse método, dentro da classe Usuario possui um processo de criptografia padrão, e para a classe professor, precisa utilizar um padrão diferente ao qual um usuário qualquer use, pra permitimos que isso seja possível utilizando a mesma assinatura de método, reescrevemos o método da classe Usuario dentro da classe Professor e adicionamos o comportamento específico e esperado daquele método para a classe Professor:

```
1 class Usuario {
2
```

```
3   public cadastrarSenha(String senha){
4     //passos e rotinas da classe Usuario
5   }
6
7 }
8
9 class Professor extends {
10
11   @Override
12   public cadastrarSenha(String senha){
13     //passos e rotinas da classe Professor
14   }
15
16 }
```

Listing 2.5: *Exemplo de polimorfismo da classe filha Professor e classe mãe Usuario*

Note que foi utilizada uma *annotation* do Java que indica que o método pertence a uma classe mãe, porém ele está sendo sobrescrito, ou seja, ele terá um comportamento diferente do padrão.

2.1.5 ORM - Object Relational Mapping

Mapeamento Objeto Relacional (ORM - Object Relational Mapping) é uma técnica que permite a persistência em um banco de dados utilizando o paradigma orientado a objeto. Isso significa que as entidades de um banco de dados podem ser representadas por meio de classes e seus atributos. A utilização dessa técnica para o desenvolvimento de um sistema computacional deve-se à facilidade que um programador tem ao desenvolver um sistema que exija uma comunicação constante com o banco de dados, pois ele não precisa obter um alto conhecimento de SQL ou banco de dados para conseguir realizar consultas, persistências, alterações dados ou qualquer tipo de ação que o banco de dados e a aplicação terão. A aplicação desse conceito no Java, é descrita com mais na seção 2.2.2.

2.1.6 Web Services

Os serviços na web é uma solução muito utilizada no mercado para permitir que diversas aplicações, independentemente da plataforma em que foram desenvolvidas, consigam se comunicar com o sistema via um protocolo de rede em comum, como por exemplo o HTTP. Essa comunicação é permitida devido ao tipo de mensagem/documento que é processado para enviar e receber os dados, como por exemplo, arquivos XML ou documentos JSON. Além de permitir a comunicação entre diversas aplicações, o web services tem o intuito de disponibilizar serviços que são acessíveis por meio de endereços públicos da aplicação.

2.1.7 Dispositivos Móveis e Aplicações

Segundo o relatório do International Data Corporation (IDC) publicado em maio de 2012, o Android possui 59% do mercado de smartphones e soma a quantia de 89,9 milhões de aparelhos distribuídos apenas no primeiro tri- mestre deste ano (2012), em todo o mundo. Em segundo lugar, aparece o iOS que é o sistema operacional do Apple iPhone. O gráfico 1.1 demonstra a par- ticipação no mercado dos principais sistemas operacionais e a quantidade de aparelhos distribuídos.

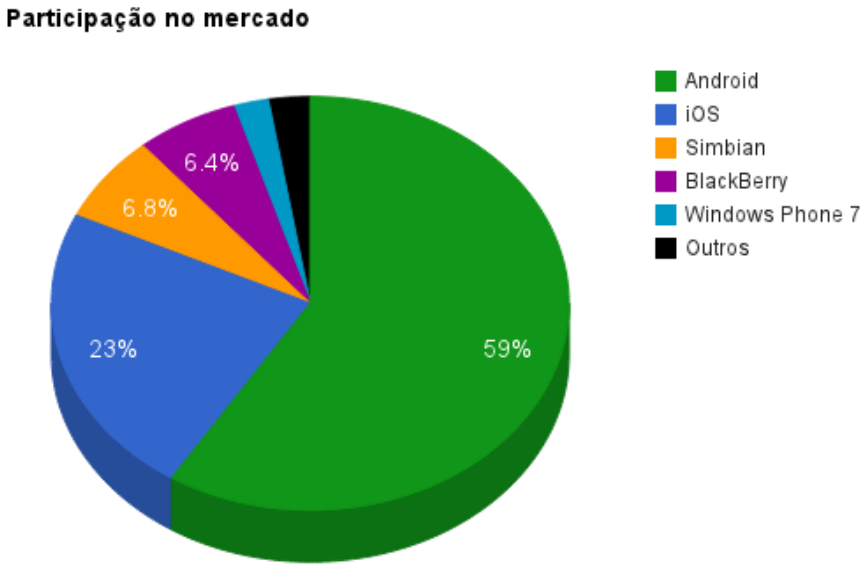

Figura 2.1: *Participação no mercado. Fonte: IDC Monteiro (2012)*

.

O IDC também prevê que, em 2016, o Android ainda possuirá a maior fatia do mercado, com 52,9%. A disputa pelo segundo lugar será acirrada entre iOS e Windows Phone 7. A 2.1 ilustra a previsão realizada pelo IDC.

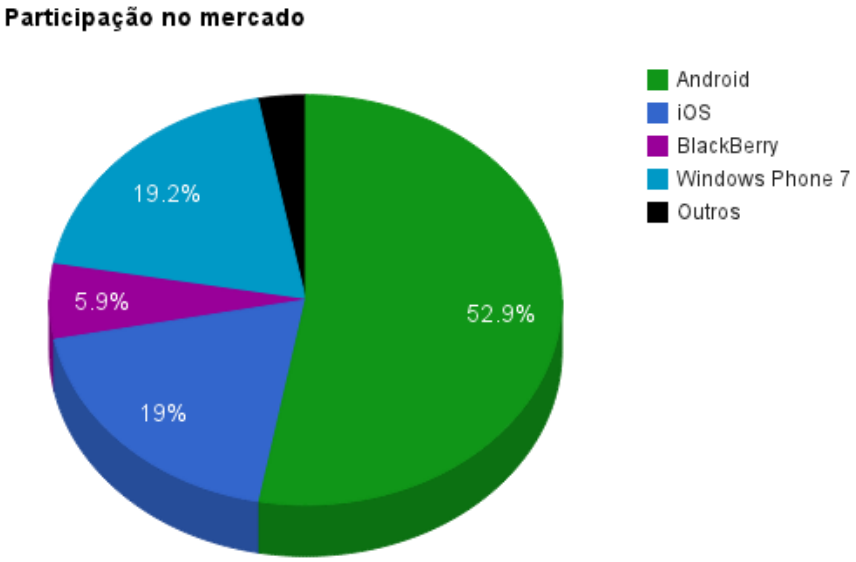

Figura 2.2: *Previsão do mercado para 2016. Fonte: IDC Monteiro (2012)*

Já no mercado de tablets, o iPad detém o trono com 68% , enquanto o Google aposta em seu primeiro tablet, o Nexus 7 ao preço US$199,00, para ga- nhar terreno tentando repetir o sucesso do Amazon Kindle Fire, que também utiliza Android.

2.1.8 MVC - Model View Controller

A arquitetura MVC consiste em dividir a aplicação em camadas:

- ***Model:*** Representa toda a regra de negócio da aplicação, ou seja, todas as classes, configuração, acessos À base de dados ou qualquer recurso que esteja do lado do servidor e que o

usuário final não tenha acesso.

- ***View:*** Todo conteúdo que representa a parte visual da aplicação, ou seja, qualquer interface com o usuário.
- ***Controller:*** É o responsável em receber todas as requisições do usuário (*View*) direcionar à aplicação o que foi solicitado (*Model*) e por fim, enviar uma resposta ao usuário.

A partir dessa arquitetura, podemos desenvolver uma aplicação de uma forma mais organizada e segura, pois além de dividirmos todo o conteúdo de acesso ao usuário (páginas em HTML ou qualquer interface direta ao usuário) e a regra de negócio (classes, configurações, acesso aos dados ou qualquer parte interna do sistema) deixamos transparente como tudo funciona internamente, ou seja, o usuário enxerga apenas o que ele precisa saber sem ter noção alguma de como é o sistema internamente.

2.1.9 TDD - Test Driven Development

O desenvolvimento das funcionalidades da aplicação, foi baseado na prática do TDD (Test Driven Development) que tem a finalidade de fazer com que o desenvolvedor crie primeiro o cenário de teste e então desenvolva a funcionalidade, por exemplo, suponhamos que precisamos criar uma funcionalidade que precise cadastrar um aluno no banco de dados, quando utilizamos o TDD criamos um método responsável por testar essa funcionalidade que a aplicação precisa.

O principal objetivo da utilização do TDD é garantir que a aplicação contenha menos bugs, pois ao invés de implementar a aplicação toda para depois testá-la, implementamos "pedaços" da aplicação e então testamos esses pedaços, ou seja, implementamos o código necessário para realizar o teste que precisamos, por exemplo, se precisamos cadastrar um aluno no sistema ou verificar se o login funciona como o esperado, implementamos cada uma dessas funcionalidades da forma mais simples possível apenas para serem testadas. A diferença entre testar pedaços da aplicação ao invés da aplicação num todo, é que permitimos que a nossa aplicação seja mais coerente, segura e flexível, pois implementaremos métodos pequenos e com apenas uma responsabilidade, dessa forma garantimos que a funcionalidade esteja funcionando e então, permitimos que ela seja reutilizada em qualquer parte do código sem preocupações de falhas. Além disso, se um dia precisarmos realizar alguma alteração de uma funcionalidade, alteraremos apenas em um ponto do sistema.

Como podemos ver, a prática do TDD, além de nos ajudar a garantir um sistema mais confiável, tende a criar diversos cenários dentro da aplicação, aumentando muito mais o desenvolvimento do sistema. A princípio parece uma prática que mais atrapalha do que ajuda o desenvolvimento, porém, quando estamos desenvolvendo, é muito comum que ocorra muitas mudanças no código e nem sempre conseguimos prever o quão impactante foi a alteração, ou seja, se ainda funciona ou se não quebrou o nosso sistema. Para verificar esse detalhe, inicialmente, pensamos em **testar manualmente tudo novamente** para verificar se ocorreu algum problema. Além de ser trabalhoso, provavelmente, esqueceremos de testar todas as funcionalidades do sistema, ou seja, a chance de não garantir que tudo esteja funcionando como o esperado, é muito grande. Quando utilizamos o TDD, todos os cenários de testes que criamos são testados automaticamente, como uma *pilha de processos*, dessa forma, evitamos perder tempo em verificar manualmente cada cenário e, se algum teste falhar, a nossa própria aplicação mostratará o que e onde falhou com muito mais precisão.

Para utizarmos o conceito de TDD em Java, podemos utilizar a API do JUnit que é descrita com mais detalhes no capítulo 2.2.8.

2.1.10 cliente e servidor

Em aplicações Web, como por exemplo websites, é muito comum encontrarmos a arquitetura de aplicação conhecida como *cliente e servidor*, que é justamente uma aplicação autônoma (servidor)

que recebe diversas requisições de diversos usuários (cliente) e responde de acordo com o que cada cliente requisitou, como por exemplo, se um cliente solicita uma página de cadastro, o servidor entende recebe essa requisição, executa os passos necessários e responde com a página solicitada.

2.1.11 Servidor de aplicação

Em aplicações Web, é muito comum utilizarmos um servidor que será responsável em manter a aplicação ativa. Esses servidores são conhecidos como servidores de aplicação e, uma de suas principais responsabilidades, é manter as aplicações disponíveis para receber requisições e respondê-las 2.1.10.

2.2 Tecnologia empregada

Está seção especifica todas as tecnologias que foram utilizadas para o desenvolvimento da aplicação, incluíndo APIs, bibliotecas e frameworks.

2.2.1 JDK - Java Development Kit

O desenvolvimento de aplicações usando a linguagem Java requer o uso do conhecido *JDK - Java Development Kit* Oracle (2015), o qual pode ser obtido gratuitamente a partir do link disponibilizado na bibliografia. Ele engloba, entre outras coisas, uma implementação da linguagem Java, um compilador, um ambiente de interpretação conhecido como máquina virtual Java etc.

2.2.2 JPA - Java Persistence API

JPA (Java Persistence API) é uma especificação Java EE que determina tudo que uma implementação de ORM (Object-Relational Mapping) para Java, precisa implementar para que seja uma biblioteca funcional de persistência de dados utilizando ORM.

2.2.3 Hibernate

O hibernate é uma das implementações da JPA 2.2.2 para o desenvolvimento de aplicações utilizando o conceito de ORM 2.1.5. Dentre as diversas implementações existentes no mercado, o hibernate é a que mais se destaca pela quantidade de desenvolvedores que o utilizam, ou seja, uma biblioteca mais estável, com maior suporte e atualizações para novas funcionalidades com mais frequência.

2.2.4 Maven

O maven é uma biblioteca capaz de gerenciar projetos Java, com a finalidade de fazer o *build* (construção) do projeto resolvendo os problemas de: gerenciamento de dependência, deploy, modularização de projeto, testes e controle de versão do projeto. O Maven permite o build do projeto utilizando o *Project Object Model* (POM). O POM é um arquivo XML que descreve todas as informaçõees que o projeto Maven possui, como por exemplo, lista de dependências, configurações de plugins, profiles entre outras iformações que o projeto possa ter. A partir do pom.xml, podemos configurar todas as propriedades para indicar como o Maven se comportará.

2.2.5 Gerenciamento de dependência

Durante o desenvolvimento da aplicação, utilizamos outras APIs para resolver determinados problemas, como por exemplo, o Spring, Hibernate, Jackson entre outros. Todas as essas APIs são arquivos .jar que precisam ser baixadas e adicionadas ao projeto, porém, quando fazemos isso manualmente, nem sempre é uma terá simples e tende apresentar problemas, como por exemplo o

Hibernate que, além de precisar das bibliotecas próprias, precisa da API da JPA. O Maven resolve todos esses problemas simplesmente adicionando a dependência por meio da tag <denpendency>:

```
<properties>
  //outras propriedades
  <hibernate.version>4.3.11.Final</hibernate.version>
</properties>

<dependencies>
  //outras dependências
  <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>${hibernate.version}</version>
  </dependency>
<dependencies>
```

Listing 2.6: *Adicionando o hibernate para a o projeto usando o pom.xml*

2.2.6 distribuição da aplicação - Deploy

Todas as vezes que terminamos uma funcionalidade ou até mesmo finalizamos a aplicação, precisamos distribuir essa aplicação para uso, ou seja, publicá-la ou, tecnicamente falando, fazer o deploy. Em muito dos casos, fazer o deploy da aplicação nem sempre é uma tarefa trivial, pois é necessário se preoculpar com ambientes diferentes, por exemplo, em ambiente de desenvolvimento, nós podemos utilizar um banco de dados de teste, podemos utilizar qualquer versão do JDK entre outras tarefas que não impactam o cliente, porém em um ambiente de produção precisamos evitar o máximo possível de bugs, ou seja, tomar cuidado com qual tipo de configuração a nossa aplicação está utilizando. Cuidar manualmente desse tipo de rotina permite um risco maior de falhas causando problemas que poderiam ser evitados, como por exemplo, utilizar um usuário e senha de banco de dados que não existe no servidor externo, e é justamente por esses motivos que podemos utilizar os *profiles* do Maven que são capazes de criar diferentes perfis para cada tipo de ambiente por meio da tag <profile> segue um exemplo de uma configuração para ambiente de desenvolvimento e para ambiente de produção:

```
<profiles>

    <profile>
      <id>Development</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        /*podemos adicionar todas as
        propriedades para ambiente de desenvolvimento*/
      </properties>
    </profile>

    <profile>
      <id>openshift</id>
      <activation>
        <activeByDefault>false</activeByDefault>
      </activation>
      <properties>
        /* podemos adicionar todas as
        propriedades para o ambiente de
        produção, nesse caso o openshift */
      </properties>
      <build>
```

Figura 2.3: *Estrutura do projeto parent*

```
25
26          <finalName>gostoudaaula</finalName>
27          <plugins>
28            <plugin>
29              <artifactId>maven-war-plugin</artifactId>
30              <version>2.1.1</version>
31              <configuration>
32                <outputDirectory>webapps</outputDirectory>
33                <warName>ROOT</warName>
34              </configuration>
35            </plugin>
36          </plugins>
37        </build>
38      </profile>
39    </profiles>
```

Listing 2.7: *Criando profiles para ambiente de desenvolvimento e ambiente de produção no pom.xml*

Além de configurar diferentes perfis da aplicação, o Maven também executa uma rotina em que verifica se todo o projeto está funcionando da maneira correta, por exemplo, verifica se todas as bibliotecas e APIs que foram descritas no pom.xml estão funcionando corretamente, faz todos os testes que foram realizados no projeto para garantir se todos funcioram e também gera os arquivos de deploy, ou seja, arquivos .jar para aplicações Java e arquivos .war para aplicações Java Web.

2.2.7 Modularização de projeto

O desenvolvimento da aplicação foi baseado em módulos, isso significa que existe mais de um projeto que resolve um problema em específico, por exemplo, existe o projeto gostoudaaula-core que possui todas as classes modelos da aplicação, o projeto gostoudaaula-db que possui todas as classes que cuidam da lógica de banco de dados e o projeto gostoudaaula-web que é a aplicação web em si. Podemos fazer uso desse tipo de arquitetura por meio dos módulos do Maven. Para criarmos módulos no Maven precisamos primeiro criar um projeto do Maven que chamando de parent, esse projeto será o projeto principal da aplicação, ou seja, iremos incluir todos os demais projetos dentro dele. Na aplicação foi criado o projeto gostoudaaula-parent. A figure 2.1 demonstra a estrutura do projeto parent:

Para adicionar módulos para esse projeto, precisaremos criar novos projetos Maven dentro desse projeto e então precisamos incluir o nome desses na tag <module>:

```
1   <modules>
2     <module>
3       gostoudaaula-core
4     </module>
5     <module>
6       gostoudaaula-web
7     </module>
8     <module>
9       gostoudaaula-db
10    </module>
11  </modules>
```

Listing 2.8: *Exemplo para associar os projetos como módulos no pom.xml*

Nesse exemplo estamos indicando que os projetos citados serão os módulos do projeto parent, porém ainda precisamos especificar aos módulos quem será o parent, então adicionamos a tag <parent> no pom.xml:

```
1  <parent>
2    <groupId>br.com.gostoudaaula</groupId>
3    <artifactId>gostoudaaula-parent</artifactId>
4    <version>//versão</version>
5  </parent>
```

Listing 2.9: *Exemplo de como informar qual é o parent do modulo no pom.xml*

2.2.8 Testes - JUnit

Como visto no capítulo 2.1.9, para implementarmos os testes em Java utilizando o conceito de TDD, utilizamos a biblioteca do JUnit que fornece anotações específicas para criarmos os cenários de teste e testá-los. Os cenários de teste são métodos com uma assinatura que descreve de uma forma bem explícita o que está sendo testado, por exemplo, um cenário de teste que verifica se a funcionalidade de cadastrar um aluno está funcionando:

```
1  @Test
2  public void deveCadastrarUmAluno(){
3    //implementação que precisa ser testada
4  }
```

Listing 2.10: *Exemplo de um caso de teste utilizando o JUnit*

Observe que esse método possui a anotação `@Test`. Essa anotação indica que esse método é testável, ou seja, no momento em que rodar o JUnit, ele fará parte da pilha de testes. Porém, para verificar o resultado de alguma funcionalidade, precisamos fazer uso dos métodos da classe `Assert` do próprio JUnit que nos fornece alguns métodos como o `assertTrue` que recebe um método com retorno *booleano*, e se caso o retorno for verdadeiro passará, caso o contrário, falhará. Podemos ver claramente nos seguintes exemplos:

```
1 import org.junit.Assert;
2 import org.junit.Test;
3
4 public class AlunoTest {
5
6   @Test
7   public void deveCadastrarUmAluno() {
8     //código para realizar o teste
9     Assert.assertTrue(cadastrarAluno());
10  }
11
12  public boolean cadastrarAluno(){
13    //implementação qualquer
14    return true;
15  }
16
17 }
```

Listing 2.11: *Exemplo não real de como utilizar o JUnit*

Observe que foi utilizado um método em que sempre retornará **true**, ou seja, o teste sempre passará, justamente para exibir a figura

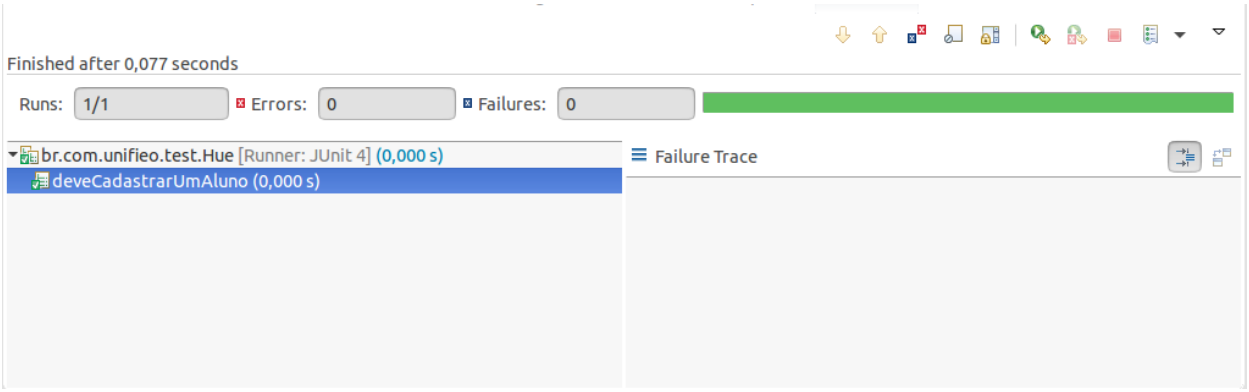

Figura 2.4: *Resultado de um teste que passou utilizando o eclipse*

.

Caso o teste falhasse, o JUnit exibiria uma informação um pouco diferente, além, de mudar a cor da barra de testes, mostraria o ponto que falhou. Podemos verificar um dos casos simplesmente alterando o método para devolver **false**:

```
1   //restante do código
2
3   public boolean cadastrarAluno(){
4     //implementação qualquer
5     return false;
6   }
7
8 }
```

Listing 2.12: *Alteração do resultado de* ***true*** *para* ***false***

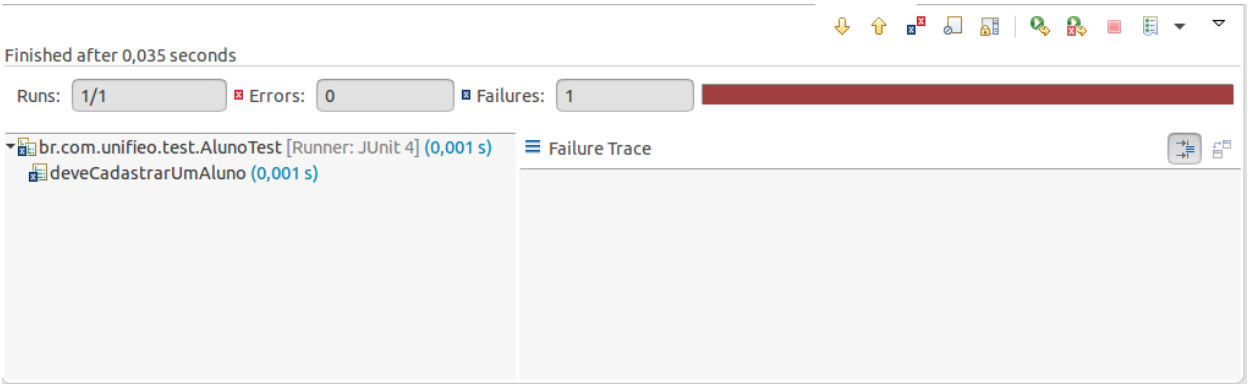

Figura 2.5: *Resultado de um teste que falhou utilizando o eclipse*

.

Perceba que além de mudar a cor para vermelho, ao clicarmos no teste *deveCadastrarUmAluno*, aparece uma informação precisa onde o testou falhou. Além disso, vemos que é mostrado o nome da classe que foi testada, o tempo que o teste levou e a quantidade de testes que foram executados.

2.2.9 Servlet API

Servlet (servidorzinho) é uma API fonercida pelo Java que permite configurarmos uma aplicação como um servidor utilizando a linguagem Java. Essas aplicações permitem uma comunicação entre cliente e aplicação utilizando o protocolo HTTP. A partir das Servlets podemos criar aplicações com o conceito de cliente e servidor 2.1.10. A API de Servlet, pode ser adquirida, gratuitamente, no site do próprio Java Servlet (2016), atualmente na versão 3.0.

2.2.10 Spring framework

O Spring framework é um conjunto de módulos (projetos) Spring (2016) em que tem a finalidade de resolver diversos problemas que são comuns em uma aplicação web, na aplicação foi utilizado para resolver as questões de controle de transações e injenção de dependência.

2.2.11 Injecão de dependência

Durante o desenvolvimento da aplicação, foi utilizado uma grande quantidade de bibliotecas e APIs, porém, em Java, ou em qualquer tipo de aplicação orientada a objetos, sempre que queremos utilizar uma classe, precisamos fazer uma instância, no caso de APIs, além de fazer uma instância, na maioria das vezes, precisamos realizar certas configurações para que funcione de acordo com a nossa necessidade. O Spring framework fornece para nós um recurso conhecido como injeção de dependência, que consiste em atribuir uma instância a um objeto automaticamente. Quando queremos que um objeto seja injetado, utilizamos a anotação @Inject, que indica que o objeto será injetado:

```
1   @Inject
2   private Aluno aluno;
```

2.2.12 Inversão de controle

A injeção de dependência atribuirá as instâncias necessárias para os objetos que estão anotados com @Inject, porém, para que o Spring consiga injetar esses objetos, precisamos ensinar o Spring a criar essas instâncias. Quando ensinamos o Spring a criar instâncias, podemos configurar como a instância será criada e então o Spring passa a gerenciar todas as classes que estão anotadas com @Inject, fazendo com que o programador não precise mais se preocupar em instânciar esses objetos, essa abordagem de fazer com que o framework instancie os objetos é chamada de Inversão de Controle (IoC).

2.2.13 Beans do Spring

Quando ensinamos o Spring a instanciar uma classe, essa classe se tornará um *Bean* do Spring. Os Beans do Spring são todas as classes gerenciadas pelo ele, ou seja, serão todas as classes que o Spring saberá criar as instâncias. Para transformarmos uma interface ou classe em um bean, basta adicionar uma tag <bean> no arquivo XML de configuração do spring:

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://
        www.springframework.org/schema/context"
3   xsi:schemaLocation="http://www.springframework.org/schema/beans
4     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
5     http://www.springframework.org/schema/context
6     http://www.springframework.org/schema/context/spring-context-3.0.xsd">
7
8   <bean id="entityManagerFactory"
9     class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
10
11  <bean id="jpaTransactionManager" class="org.springframework.orm.jpa.
        JpaTransactionManager">
12    <property name="entityManagerFactory" ref="entityManagerFactory" />
13  </bean>
14
15  //para adicionar mais beans, adicione mais tags <bean> e ensine como a classe
16  precisa ser instanciada
17
18 </beans>
```

2.2.14 Controle de transações

Para conectar a aplicação com o banco de dados, foi utilizada a especificação JPA (Java Persistence API) que estabalece uma comunição com o banco de dados criando transações. Porém, todas as vezes que estamos utilizando uma transação precisamos saber o momento em que ela precisa ser iniciada e o momento em ela precisa aplicar as ações de commit/rollback e por fechá-las. Perceba que essa ações podem ser um pouco repitivas como também perigosas para a nossa aplicação, como por exemplo, interagir com o banco por meio de uma transação e não realizar o commit, ou então, simplesmente não fechar a conexão. Devido a esses detalhes, utilizamos o Spring para fazer a injeção de dependência das classes que são responsáveis em lidar com as transações, ou seja, o Spring gerencia todas as transações da aplicação, garantindo que todas essas rotinas sejam feitas corretamente sem o desenvolvedor precisar se preocupar.

2.2.15 Jackson

Em uma comunicação entre aplicação utilizando Web Services 2.1.6, é muito comum convertemos classes para um formato em que qualquer aplicação, independemente da implementação, ou seja, linguagem de programação, entenda, como por exemplo, XML e JSON. Porém, uma conversão manual de uma classe passar o formato desejado, nem sempre é um trabalho fácil e por isso não é recomendável esse tipo de prática. Para facilitarmos essa tarefa, utilizamos a biblioteca do jackson que, com poucas configurações, consegue realizar essas conversões de forma automática.

2.2.16 Openshift

O openshift é uma plataforma como um serviço, também conhecido como PaaS (Platform-as-a-Service), da Red Hat que permite o os desenvolvedores a criar aplicações em uma infraestrutura na nuvem. Para mais imformações da plataforma consulte o web site Openshift (2016).

2.2.17 Eclipse IDE for Java EE

Integrated Development Environment (ambiente de desenvolvimento integrado) são ferramentas que auxiliam o desenvolvedor durante o desenvolvimento de uma aplicação. O propósito das IDEs é justamente minimizar o tempo de desenvolvimento de um projeto, como por exemplo, auto preenchimento de código, busca de recursos, compilação instantânia, sugestão para melhores implementações, realização de testes entre outras características comuns entre as principais IDEs do mercado. O Eclipse é uma IDE IDE (2016) gratuita e pode ser adquirida no link disponibilizado na bibliografia. Existem diversas versões para diversos fins, porém, para essa aplicação, foi utilizada a versão for Java EE que é justamente predefinida para o desenvolvimento de aplicações Java Web.

2.2.18 Android Studio

Assim como o Eclipse IDE 2.2.17, o Android Studio é uma IDE, porém a sua finalidade é justamente facilitar o desenvolvimento de aplicações para Android. Com diversas características, funcionalidades e recursos para facilitar o desenvolvimento de uma app. Da mesma forma que o Eclipse IDE, o Android Studio Studio (2016) é gratuito e pode ser adquirido no link disponibilizado na bibliografia.

Capítulo 3

Projeto Técnico

3.1 Modelagem de banco de dados

A aplicação *Gostou da Aula?* teve seu desenvolvimento iniciado pela modelagem conceitual dos dados. Após análise detalhada, obtivemos um Diagrama Entidade Relacionamento, o qual é exibido pela figura 3.1.

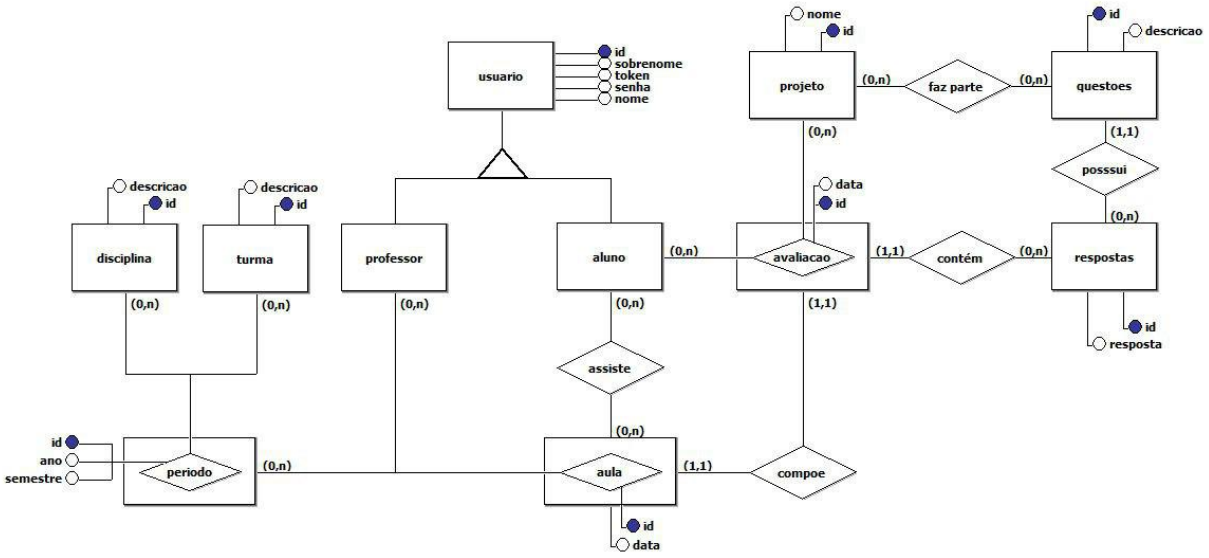

Figura 3.1: *Diagrama Entidade Relacionamento - DER*

.

Cada entitidade e relacionamento possui determinadas peculiaridades que são descritas a seguir:

- **Usuário**: Refere a uma representação de um usuário qualquer no sistema, essa é uma entidade abstrata, ou seja, exige uma especilização entre a entidade **Professor** e **Aluno** que são os 2 tipos de usuários existentes na aplicação.
- **Professor**: Representa um professor dentro da aplicação.
- **Aluno**: Representa um aluno dentro da aplicação.
- **Aula**: Representa uma aula dentro da aplicação, é constituída por **Aluno(os)**, um **Professor** e um **Periodo**.
- **Periodo**: Representa um período letivo dentro da instituição, é constituído por uma **Disciplina** e uma **Turma**.

- **Turma**: Representa uma turma da instituição.
- **Disciplina**: Representa uma disciplina da instituição.
- **Avaliacao**: Representa uma avaliação dentro da aplicação, é constituída por uma **Aula**, um **Projeto** e **Resposta(as)**.
- **Projeto**: Representa um projeto dentro da aplicação constituído por **Questoes** das avaliações.
- **Questoes**: Representa todas as questões de um determinado **Projeto**. **Respostas**: Representa todas as respostas de **Questoes** e **Avaliacao**.

3.2 Criação do projeto via Maven

Para a criação do projeto, foi utilizada a biblioteca do Maven 2.2.4. Para criarmos um projeto utilizando o maven, podemos utilizar o Eclipse IDE for Java EE.

Capítulo 4

Relação com disciplinas do curso de Ciência da Computação do UNIFIEO

Aqui você vai citar cada disciplina que cursou e dizer qual a relação que ela teve com o desenvolvimento desse projeto.

Capítulo 5

Conclusões

Aqui vamos colocar algumas conclusões. Vamos pensar nelas lá no fim do semestre, quando o projeto já estiver quase pronto. Provavelmente vamos falar das observações do prof. Fernando sobre as questões e ética aqui, sobre o quanto foram importantes para a elaboração do questionário.

Capítulo 6

Trabalhos Futuros

Aqui vamos colocar sugestões de trabalhos futuros.

Apêndice A

Dicionário de Dados

Aqui vamos colocar o dicionário de dados.

Referências Bibliográficas

Aniche (2012) Maurício Aniche. *Test-Driven Development*. Casa do Código, 1º ed. Citado na pág.

Bates e Sierra (2005) Bert Bates e Kathy Sierra. *Use a Cabeça! Java*. Altabooks Editora, 2º ed. Citado na pág.

Coelho (2013) Hébert Coelho. *JPA Eficaz*. Casa do código, 1º ed. Citado na pág.

Deitel e Deitel (2010) Paul Deitel e Harvey Deitel. *Java Como Programar*. Pearson Education, 8º ed. Citado na pág.

Deitel *et al.* (2012a) Paul Deitel, Harvey Deitel, Abbey Deitel e Michael Morgano. *Android For Programmers An App Driven Approach*. Pearson Education, 1º ed. Citado na pág.

Deitel *et al.* (2012b) Paul Deitel, Harvey Deitel, Abbey Deitel e Michael Morgano. *Android Para Programadores Uma Abordagem baseada em aplicativos*. Bookman, 1º ed. Citado na pág.

Google (2015) Google. Android developers. http://developer.android.com/index.html, May 2015. Último acesso em 05/05/2015. Citado na pág.

IDE (2016) Eclipse IDE. Eclipse ide for java ee. http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2, April 2016. Último acesso em 12/04/2016. Citado na pág. 16

Lecheta (2013) Ricardo Lecheta. *Google Android Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Novatec, 3º ed. Citado na pág.

Monteiro (2012) João Bosco Monteiro. *Google Android*. Casa do Código, 1º ed. Citado na pág. ix, 8

Openshift (2016) Openshift. Openshift red hat. https://www.openshift.com/, April 2016. Último acesso em 11/04/2016. Citado na pág. 16

Oracle (2015) Oracle. Java se - downloads | oracle technology network | oracle. http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html, January 2015. Último acesso em 05/01/2016. Citado na pág. 10

Sanderson (2012) Daniel Sanderson. *Programming Google App Engine*. O'Reilly Media, 2º ed. Citado na pág.

Servlet (2016) Servlet. Servlet api 3.0. http://download.oracle.com/otndocs/jcp/servlet-3.0-fr-eval-oth-JSpec/, April 2016. Último acesso em 11/04/2016. Citado na pág. 14

Spring (2016) Spring. Spring projects. https://spring.io/projects, April 2016. Último acesso em 10/04/2016. Citado na pág. 15

Studio (2016) Android Studio. Android studio. http://developer.android.com/intl/pt-br/sdk/index.html, April 2016. Último acesso em 12/04/2016. Citado na pág. 16