

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

THIAGO DE AZEVEDO DORNELLES

**Reconstrução do software AvalWeb®
usando conceitos de SPA, REST e NoSQL**

Monografia apresentada como requisito parcial
para a obtenção do grau de Bacharel em Ciência
da Computação

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2016

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Rui Vicente Oppermann

Vice-Reitora: Prof^a. Jane Fraga Tutikian

Pró-Reitor de Graduação: Prof. Vladimir Pinheiro do Nascimento

Diretora do Instituto de Informática: Prof^a. Carla Maria Dal Sasso Freitas

Coordenador do Curso de Ciência de Computação: Prof. Sérgio Luis Cechin

Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Gostaria de agradecer aos meus pais que sempre estiveram ao meu lado me apoiando na realização de qualquer objetivo que eu já tenha tentado atingir, pela paciência, pelo cuidado, pelo amor recebido todos os dias. À minha namorada que me apoiou de forma ativa neste e em outros trabalhos, me incentivando na busca deste objetivo, sempre amorosa e quando possível sendo paciente. Aos meus familiares que mesmo não estando tão próximos me fazem uma pessoa mais alegre. Aos amigos pelos momentos de compreensão pela ausência. Ao meu avô Olávio (*In Memoriam*), que infelizmente não estará em corpo conosco para comemorar este momento e finalmente ao Prof. Leandro Wives pela disposição em ajudar na realização deste trabalho, sempre propondo ideias para realizar um trabalho melhor.

RESUMO

A produção de softwares voltados para educação teve um grande impulso nos últimos anos devido ao crescente aumento do acesso à internet, possibilitando a colaboração de milhares de pessoas pelo mundo. Iniciativas variadas compõem um leque de opções para aprendizagem na web, com cursos online abertos e massivos (MOOCs), repositórios de objetos de aprendizagem, avaliação e publicação de conteúdo. Especificamente na área da avaliação, existe o AvalWeb[®], um software que permite gerenciar questões possibilitando a condução de uma avaliação de forma dinâmica com o objetivo de aferir o conhecimento do aluno com maior precisão. Este trabalho descreve as tecnologias e o projeto de implementação de uma nova versão do sistema AvalWeb[®] que o torna aderente às novas tecnologias para a web, conferindo-lhe a responsividade e a facilidade de uso dos modernos web Apps.

Palavras-chave: AngularJS. SPA. MongoDB. NoSQL. Avaliação Online.

AvalWeb® Software reconstruction using concepts of SPA, REST and NoSQL

ABSTRACT

The education software production had a major boost in recent years because of increasing access to the internet and the possibility of collaboration of thousands of people around the world. Several initiatives opens up a wide array of options for learning on the web, with massive open online courses (MOOCs), learning objects repositories, evaluation and content publication. Specifically in the evaluation field, there is the AvalWeb® application, a software that allows management of questions, creating an evaluation that establishes dynamic rules that can make a test measure the student's knowledge more accurately. This paper describes the technology and the design of a new version of the AvalWeb® system using new technologies for the web that provides responsive pages and ease of use of modern web Apps.

Keywords: AngularJs, SPA, MongoDB, NoSQL, Online Evaluation.

LISTA DE FIGURAS

Figura 2.1	Tela de cadastro de questões do AvalWeb® Alpha	14
Figura 2.2	Diagrama representativo de funcionalidades do AvalWeb® Alpha	14
Figura 2.3	Exemplo de feedback do AvalWeb® 2.0.....	16
Figura 3.1	Exemplo de um array de objetos usando JSON.....	22
Figura 3.2	Exemplo de uma aplicação utilizando Angular Material	24
Figura 4.1	Diagrama das entidades da base de dados NoSQL.....	37
Figura 4.2	Diagrama de navegação do sistema AvalWeb®	43
Figura 5.1	Configuração de pacote NPM do sistema AvalWeb®	46
Figura 5.2	Função de autenticação do sistema.....	47
Figura 5.3	Roteador simples de método GET.....	48
Figura 5.4	Schema da entidade Student e modelo gerado a partir do schema.....	49
Figura 5.5	Estratégia balanço de dificuldade	50
Figura 6.1	Botão de recolhimento de menu	53
Figura 6.2	Ícone básicos do sistema.....	53
Figura 6.3	Tela de autenticação do AvalWeb®	54
Figura 6.4	Tela de edição de questões.....	55
Figura 6.5	Tela de hierarquia de categorias.....	56
Figura 6.6	Tela de cadastro de turmas.....	57
Figura 6.7	Convites por CSV	57
Figura 6.8	Cadastro de avaliação por categorias.....	59
Figura 6.9	Relatório de avaliações	60

LISTA DE TABELAS

Tabela 2.1	Comparativo das versões do AvalWeb® quanto as funcionalidades.....	17
Tabela 2.2	Comparativo de versões quanto a características não-funcionais.....	17
Tabela 4.1	Tabela de comandos para o recurso questions.....	38
Tabela 4.2	Tabela de comandos para o recurso categories.....	39
Tabela 4.3	Tabela de comandos para o recurso students.....	39
Tabela 4.4	Tabela de comandos para o recurso classes.....	40
Tabela 4.5	Tabela de comandos para o recurso users.....	40
Tabela 4.6	Tabela de comandos para o recurso tests.....	41
Tabela 4.7	Tabela de comandos para o recurso studenttests.....	42
Tabela 4.8	Tabela de comandos para o recurso invites.....	42
Tabela 4.9	Tabela de comandos para o recurso reports.....	43

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CSS	Cascading Style Sheets
CSV	Comma-Separeted Values
DOM	Document Object Model
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
NoSQL	Not Only Structured Query Language
REST	Representational State Transfer
SPA	Single Page Application
XHR	XML HTTP Request
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO	11
1.1 Objetivo.....	11
1.2 Estrutura do texto	11
2 TRABALHOS RELACIONADOS	13
2.1 AvalWeb® Alpha.....	13
2.2 AvalWeb® 1.0	14
2.3 AvalWeb® 2.0	15
2.4 Comparativo das versões do AvalWeb®	16
2.5 Funcionalidades novas.....	17
2.5.1 Hierarquia de categorias	18
2.5.2 Convite de aluno por e-mail.....	18
2.5.3 Interface responsiva	18
2.5.4 Interoperabilidade via API REST	19
2.5.5 Estratégias de prova dinâmica.....	19
3 CONCEITOS E TECNOLOGIAS UTILIZADAS	20
3.1 Front-end	20
3.1.1 HTML e CSS	21
3.1.2 JavaScript	21
3.1.3 AJAX.....	21
3.1.4 JSON.....	22
3.1.5 SPA.....	22
3.1.6 AngularJS.....	23
3.1.7 Angular Material.....	24
3.2 Back-End	25
3.2.1 Node.js	25
3.2.2 Sistema de pacotes NPM	25
3.2.3 REST.....	26
3.2.4 Express.....	27
3.2.5 Passport.....	27
3.2.6 MongoDB	27
3.2.7 Mongoose.....	28
4 PROJETO DA APLICAÇÃO	30
4.1 User Stories	30
4.1.1 Stories do professor.....	30
4.1.2 Stories do aluno.....	31
4.2 Entidades	32
4.2.1 Questions.....	32
4.2.2 Categories	33
4.2.3 Students.....	33
4.2.4 Classes.....	34
4.2.5 Users	34
4.2.6 Tests	35
4.2.7 Student Tests	35
4.2.8 Diagrama de entidades	36
4.3 API REST	37
4.3.1 Questions.....	38
4.3.2 Categories	39
4.3.3 Students.....	39

4.3.4	Classes.....	40
4.3.5	Users	40
4.3.6	Tests	41
4.3.7	Student Tests	42
4.3.8	Invites.....	42
4.3.9	Reports	43
4.4	Diagrama de navegação.....	43
5	IMPLEMENTAÇÃO DO PROJETO.....	45
5.1	Configuração de projeto NPM.....	45
5.2	Autenticação	46
5.3	Organização dos componentes do Sistema	47
5.3.1	Roteadores do Framework Express.....	47
5.3.2	Schemas do framework Mongoose	48
5.3.3	Páginas estáticas com AngularJS.....	49
5.4	Padrão Strategy e estratégias de aplicação de avaliação.....	50
6	GUIA DE USO DO SISTEMA	52
6.1	Interface do sistema	52
6.1.1	Ícones de botões do sistema.....	52
6.2	Atividades do professor	54
6.2.1	Autenticação	54
6.2.2	Cadastro de questões.....	55
6.2.3	Cadastro de categorias	55
6.2.4	Cadastro de alunos	56
6.2.5	Cadastro de turmas.....	56
6.2.6	Convite de alunos.....	57
6.2.7	Cadastro de avaliações	58
6.2.8	Relatório de avaliações	59
6.3	Atividades do aluno.....	60
7	CONCLUSÃO E TRABALHOS FUTUROS	61
7.1	Aspectos técnicos.....	61
7.2	Trabalhos futuros.....	62
	REFERÊNCIAS.....	63

1 INTRODUÇÃO

O AvalWeb[®] é um sistema que propõe a realização de avaliações eletrônicas com objetivo de gerar informações sobre os alunos e o seu desempenho, apoiando o educador no processo de ensino. Foi desenvolvido inicialmente em dissertação de mestrado (CAR-DOSO, 2001) e posteriormente refinado em monografia de conclusão de curso (TORRES, 2009). As suas implementações incrementaram funcionalidades e deixaram sugestões de expansão para trabalhos futuros.

A última versão do AvalWeb[®] data do ano de 2009 (TORRES, 2009), e desde então não houve mais atualizações na sua estrutura. A proposta de uma nova arquitetura se faz necessária para que o sistema se comporte adequadamente nos diferentes dispositivos móveis da atualidade. Ainda percebeu-se a oportunidade de fazer a reengenharia da sua camada de back-end ao implementar uma arquitetura baseada em *REST*, permitindo um melhor desempenho e interoperabilidade com outros sistemas que possam vir a integrá-lo.

Adicionalmente, esta versão do sistema foi projetada para que fosse possível ter versatilidade no uso de estratégias de aplicação de avaliações dinâmicas.

1.1 Objetivo

O objetivo deste trabalho é a reconstrução do AvalWeb[®], criando uma aplicação web de página única (SPA) com uma interface intuitiva, responsiva e funcional, que é capaz de funcionar tanto em dispositivos móveis como em computadores pessoais, permitindo mudanças no projeto com facilidade e que seja extensível para agregar novas funcionalidades.

Este trabalho tem também o objetivo de apresentar o projeto e as tecnologias envolvidas neste processo de reconstrução, como AngularJS e Angular Material na camada de Front-End e as tecnologias usadas na camada de Back-End: Node.js, Express e MongoDB.

1.2 Estrutura do texto

Este trabalho está dividido em 7 capítulos. O capítulo 1 apresenta o sistema AvalWeb[®], contextualiza a sua aplicação e apresenta os objetivos deste trabalho. O capí-

tulo 2 apresenta as versões anteriores e a atual do sistema, fazendo um comparativo entre as funcionalidades. Ao final apresenta as funcionalidades incluídas na versão produzida neste trabalho. O capítulo 3 explica os conceitos e tecnologias empregadas no projeto da nova versão do AvalWeb[®], classificando os assuntos em duas grandes seções, Front-end e Back-end. O capítulo 4 fala sobre o desenvolvimento do projeto da aplicação, mostrando os requisitos, entidades, arquitetura e navegação do sistema. O capítulo 5 mostra detalhes da implementação do sistema e as características dos frameworks utilizados. O capítulo 6 apresenta um guia de uso para o AvalWeb[®]. Por fim no capítulo 7 são apresentadas as conclusões, aspectos técnicos envolvidos para construção do sistema e sugestões para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Este capítulo introduz os trabalhos acadêmicos relacionados ao AvalWeb® e suas principais funcionalidades implementadas. Primeiramente na seção 2.1 é apresentado o AvalWeb® Alpha, a primeira versão do sistema. Em seguida na seção 2.2, é descrito o AvalWeb® 1.0 e suas melhorias em relação à versão anterior. Na seção 2.3 são descritos os avanços de projeto da última versão antes deste trabalho. Na seção 2.4 são comparadas as versões anteriores com a versão proposta neste trabalho. Na seção 2.5 são explicadas as novas funcionalidades implementadas.

2.1 AvalWeb® Alpha

Esta versão inicial denominada AvalWeb® Alpha pelo trabalho de graduação de Dalton de Oliveira Torres (TORRES, 2009) foi desenvolvida na dissertação de mestrado de Rodrigo Ferrugem Cardoso em 2001 (CARDOSO, 2001) e era um sistema para armazenamento de questões e avaliações para aplicação na web. Basicamente o que o sistema tinha implementado era um repositório de questões do qual se poderia gerar avaliações para serem respondidas pelos alunos.

O cadastro de questões implementado possuía diversas opções para escolha do estilo de questão, sendo possível fazer, por exemplo, questões de múltipla escolha, marcação verdadeiro e falso, relacionar as colunas e resposta dissertativa. Na figura 2.1 podemos ver a interface para o cadastro de questões.

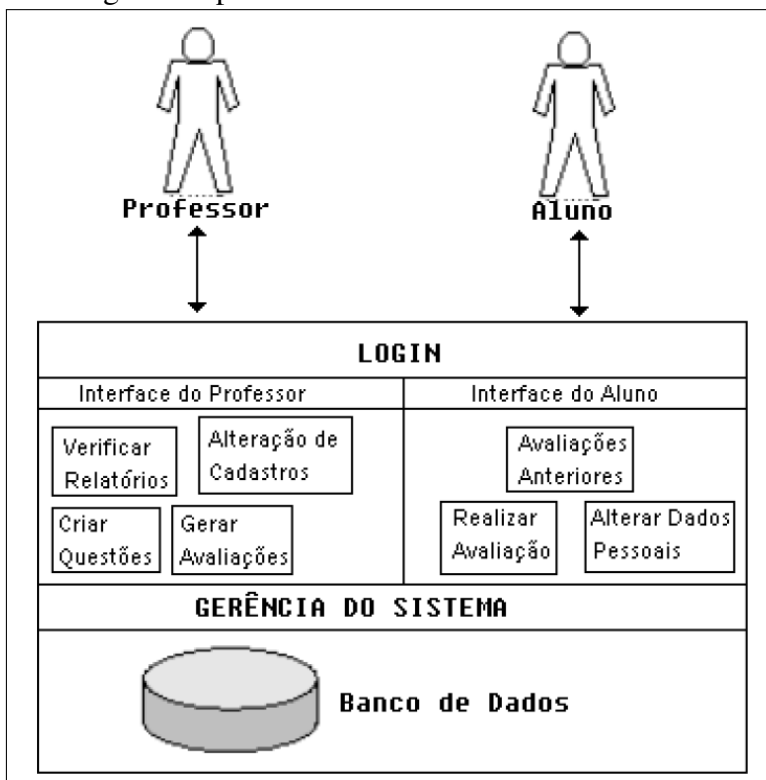
O AvalWeb® Alpha trabalhava com apenas dois atores operando o sistema, o professor e o aluno. O professor tinha acesso à todas as funcionalidades, tendo o papel de administrador dentro do sistema, cadastrando alunos, disciplinas e turmas. O aluno tinha acesso aos seus dados e as avaliações, respondendo e verificando os resultados. Na figura 2.2 temos uma diagrama que mostra a organização do sistema.

O AvalWeb® Alpha possuía uma interface baseada no estilo de sites produzidos à época de sua criação, sendo construído como um site dinâmico baseado na atualização de conteúdo através do redirecionamento entre páginas.

Figura 2.1: Tela de cadastro de questões do AvalWeb® Alpha

Fonte: Cardoso (2001).

Figura 2.2: Diagrama representativo de funcionalidades do AvalWeb® Alpha



Fonte: Cardoso (2001).

2.2 AvalWeb® 1.0

Na sua versão 1.0 o AvalWeb® passou a agregar mais funcionalidades com o objetivo de permitir que o professor obtenha informações estatísticas sobre os alunos para

que possa melhorar o ensino da classe. Algumas dessas funcionalidades foram: marcação do tempo médio de resposta para as questões, média de acertos e erros em cada questão, sugestões automáticas para alteração de dificuldade de questões (MORAIS; LIMA; FRANCO, 2005).

Essas funcionalidades já começam a definir o novo objetivo do sistema, que é de avaliar e ensinar simultaneamente (TORRES, 2009), além de fornecer ao professor dados importantes para uma análise do desempenho da turma.

Outras funcionalidades que também já eram citadas neste trabalho, mas não foram implementadas na ocasião foram o feedback por questão, que tinha o objetivo de exibir comentários para o aluno conforme a questão respondida, e o comentário do professor ao final da prova, orientando o aluno sobre as questões erradas pelo aluno (TORRES, 2009).

2.3 AvalWeb® 2.0

Na versão 2.0 foi desenvolvida em trabalho de conclusão de graduação no ano de 2009 por Dalton de Oliveira Torres e trouxe mudanças estruturais na aplicação, com a modularização do sistema em quatro módulos independentes com comunicação entre si. Os módulos ficaram divididos em: banco de questões, banco de provas, aplicação e banco de retorno (TORRES, 2009).

A aplicação foi feita utilizando PHP, HTML e banco de dados MySQL, implementando uma parte das ideias dos trabalhos anteriores, como por exemplo, as provas dinâmicas.

As provas dinâmicas são provas elaboradas apenas selecionando os assuntos que devem ser perguntados na avaliação, sendo a prova totalmente guiada conforme os acertos ou erros do aluno na resolução das questões. Nesta modalidade de prova as questões são apresentadas uma de cada vez, e de acordo com os acertos e erros do aluno são encaminhadas novas questões mais fáceis ou difíceis (TORRES, 2009).

Ainda nesta versão foi implementado o modo aprendizado, uma opção que quando habilitada na avaliação, permite que o aluno tenha um feedback de questão imediatamente após respondê-la expondo o aluno ao conteúdo durante a prova. Na figura 2.3 podemos ver um exemplo de feedback sobre uma resposta de questão.

Figura 2.3: Exemplo de feedback do AvalWeb® 2.0



Fonte: Torres (2009).

2.4 Comparativo das versões do AvalWeb®

No trabalho sobre o AvalWeb® 2.0 (TORRES, 2009), é feito um comparativo de funcionalidades entre as versões do AvalWeb®. Nessa comparação não foram levados em conta requisitos não-funcionais daquelas versões. Além disso, os modelos de páginas web da época tinham requisitos funcionais e não-funcionais diferentes dos da atualidade, pois hoje os navegadores estão mais padronizados e suportam uma gama maior de possibilidades de programação para a interação com o usuário.

Com esse contexto faremos uma comparação das funcionalidades entre as versões anteriores e a desenvolvida neste trabalho, para depois isoladamente apresentarmos as características não-funcionais que constituem as principais melhorias deste trabalho.

A tabela 2.1 é parcialmente baseada na tabela criada no trabalho de (TORRES, 2009) com o acréscimo das funcionalidades inseridas nesta implementação que será denominada aqui *AvalWeb® SPA*.

Tabela 2.1: Comparativo das versões do AvalWeb[®] quanto as funcionalidades

Descrição	AvalWeb [®] Alpha	AvalWeb [®] 1.0	AvalWeb [®] 2.0	AvalWeb [®] SPA
Tipos de questões	✓	✓	✓	
Dificuldade da questão	✓	✓	✓	✓
Definir a dificuldade da próxima questão			✓	
Feedback por questão		✓	✓	
Feedback por alternativa			✓	✓
Feedback por prova	✓	✓	✓	
Banco de questões	✓	✓	✓	✓
Hierarquia de assuntos (categorias)				✓
Questões com imagens			✓	✓
Histórico das provas			✓	✓
Convite de alunos por e-mail				✓

A tabela 2.2 mostra as melhorias não-funcionais atingidas pela nova versão proposta neste trabalho.

Tabela 2.2: Comparativo de versões quanto a características não-funcionais

Descrição	AvalWeb [®] Alpha	AvalWeb [®] 1.0	AvalWeb [®] 2.0	AvalWeb [®] SPA
Acesso via web	✓	✓	✓	✓
Interface responsiva				✓
Interoperabilidade via API REST				✓
Estratégias de prova dinâmica				✓

Nas tabelas acima observa-se que a versão atual se destaca principalmente pelas características que a torna mais moderna e acessível pelos computadores e dispositivos móveis diversos.

2.5 Funcionalidades novas

Apresenta-se aqui as funcionalidades que foram mostradas nas tabelas 2.1 e 2.2 e que não foram implementadas nas versões anteriores do AvalWeb[®].

2.5.1 Hierarquia de categorias

As categorias de questões ajudam na elaboração de provas de forma mais rápida, permitindo ao professor selecionar apenas as categorias de assuntos que gostaria de abordar na prova para gerá-la.

Essa funcionalidade demanda que o professor classifique as questões em categorias e que principalmente, organize-as hierarquicamente para que o sistema possa buscar com maior precisão os assuntos abordados.

Por exemplo, se um professor deseja criar uma prova de matemática sobre aritmética, mas quer ter certeza que a prova irá cobrar assuntos sobre multiplicação, então o professor poderá criar uma subcategoria da categoria matemática, chamada multiplicação. Ao criar uma avaliação ele poderá selecionar a categoria de multiplicação, para que na prova seja cobrada uma questão deste assunto sem a necessidade da escolha direta de uma questão.

Se o professor preferir, poderá selecionar uma questão de um assunto mais genérico, selecionando uma categoria de nível mais alto. Assim, o aluno poderá receber uma questão sobre a categoria selecionada ou sobre as subcategorias abaixo desta.

2.5.2 Convite de aluno por e-mail

Os convites por e-mail facilitam o processo de cadastro de alunos no sistema, pois através de uma lista de e-mails podemos gerar e encaminhar os dados de acesso para uma grande quantidade de alunos.

No convite por e-mail a entrada de dados utilizada é uma lista em formato *CSV*, ou seja, uma lista de nomes de alunos e e-mails separados por vírgula. Este formato é gerado na maioria dos clientes de e-mails atuais, o que facilita a obtenção dos dados para gerar uma turma.

2.5.3 Interface responsiva

Uma interface web responsiva almeja permitir que uma página seja vista em função do tamanho do dispositivo que a está visualizando (MARCOTTE, 2010). A interface responsiva permite que o AvalWeb® tenha o seu conteúdo exibido de forma melhor dis-

tribuída em cada tamanho de tela dos dispositivos móveis.

2.5.4 Interoperabilidade via API REST

A API REST permite que outros programadores possam desenvolver softwares que trocam dados e realizam ações sobre o back-end do AvalWeb®. Por exemplo, se for desenvolvido um aplicativo móvel para acesso ao sistema, não será necessária a reprogramação do código que roda no servidor.

2.5.5 Estratégias de prova dinâmica

As estratégias de prova são algoritmos que selecionam as próximas questões durante a prova. Estes algoritmos podem ser personalizados pelo programador com mais facilidade nesta versão.

Foi implementada uma classe no sistema que pode ser estendida para que o AvalWeb® receba mais opções de estratégias. Assim, o programador poderá adicionar novas formas de encaminhar questões apenas adicionando um novo arquivo de código-fonte.

3 CONCEITOS E TECNOLOGIAS UTILIZADAS

Este capítulo está dividido em duas seções que definem os componentes e as características das duas camadas do sistema (Front-end e Back-end) e que em suas subseções apresentam seus conceitos e tecnologias.

A primeira seção trata da apresentação dos conceitos básicos relacionados às tecnologias empregadas na construção da camada Front-end do sistema, ou seja, àquelas tecnologias que são executadas no lado cliente da aplicação web, como a linguagem de marcação de páginas web *HTML*, a definição de estilos em cascata *CSS*, a linguagem *Javascript* e o framework *AngularJS*.

A segunda seção apresenta os conceitos e tecnologias que compõem o Back-end do sistema, ou seja, as tecnologias que rodam do lado servidor da aplicação, como por exemplo o banco de dados *MongoDB*, a plataforma de aplicações *Javascript Node.js* e o framework *Express*.

3.1 Front-end

Nesta seção apresenta-se o conceito de Front-end em uma aplicação web. Nas subseções apresenta-se as tecnologias utilizadas para construção deste Front-end.

O Front-end de um aplicativo web contém todos os componentes que trabalham no lado cliente da aplicação. Esta camada é desenvolvida usando tecnologias como HTML, CSS, Javascript, AJAX, AngularJS, Angular Material e JSON (TABLELESS, 2012).

Neste projeto, a interface com o usuário tem a responsabilidade de apresentar os elementos visuais que permitem a entrada de dados, a renderização dos dados junto aos controles da interface e o processamento dos dados recebidos e enviados ao Back-End do sistema.

Nesta versão do AvalWeb[®], o Front-end realiza uma boa parte das atribuições do sistema, deixando menos responsabilidades para o Back-end do que as páginas web convencionais. Tarefas como a renderização dos dados no browser, roteamento dos links internos da página, cálculos matemáticos, interpretação de dados estruturados e algumas regras de negócio são executados pelo browser.

3.1.1 HTML e CSS

HTML é uma linguagem de marcação na qual as páginas web são construídas. Seu nome é uma abreviação de HyperText Markup Language. Esta linguagem é interpretada por todos navegadores de internet modernos e é usada atualmente para estruturar uma página, definindo também a semântica dessa estrutura. Trabalhando em conjunto com o CSS (Cascading Style Sheets), permite a definição da aparência dos elementos da página, permitindo separar a apresentação visual da estrutura básica do documento (W3C, 2016). Assim, pode-se dizer que uma página HTML pode ter infinitas aparências, dependendo apenas da criação de um novo documento CSS que o decore.

3.1.2 JavaScript

JavaScript é uma linguagem de programação interpretada (FLANAGAN, 2002), originalmente implementada nos navegadores web para uso em scripts que rodam no lado do cliente. Atualmente é a principal linguagem para programação usada para interação dos elementos de páginas web.

Nos últimos anos o Javascript começou a ser bastante utilizado no lado do servidor de aplicação, através do ambiente Node.js (OVERFLOW, 2016).

É uma linguagem bastante versátil, podendo implementar características de diversos paradigmas de programação, como orientação à objetos e programação funcional (MOZILLA, 2015).

Neste projeto o Javascript é a linguagem que está presente em quase todas as partes do sistema. Tanto na camada de Front-end quanto na camada de Back-end a linguagem usada é a mesma, o que pode ser uma vantagem para o projeto por não necessitar de conhecimentos sobre outras linguagens como PHP, Python e Java que são geralmente utilizadas no Back-end dos sistemas web.

3.1.3 AJAX

AJAX é um acrônimo para Asynchronous JavaScript and XML. Basicamente, AJAX é o uso do objeto XMLHttpRequest para comunicação assíncrona com lado servidor.

A natureza assíncrona das requisições feitas pelo uso do AJAX permite a criação de páginas que atualizam suas informações sem a necessidade de recarregamento completo da página (MOZILLA, 2016).

3.1.4 JSON

JSON é o acrônimo para Javascript Object Notation. É um formato de dados leve para intercâmbio de informações entre sistemas. Este formato facilita o reconhecimento e geração de dados. Tem a sua sintaxe baseada em um subconjunto de instruções da linguagem Javascript, tendo sua representação de dados formada pelas estruturas existentes na linguagem (JSON.ORG, 2016).

Na figura 3.1 apresenta-se o exemplo de uma lista de objetos que representam pessoas com seus nomes e idades em formato JSON.

Figura 3.1: Exemplo de um array de objetos usando JSON

```
[
  {
    "name": "Pedro",
    "age": 29
  },
  {
    "name": "Clara",
    "age": 23
  },
  {
    "name": "Silvio",
    "age": 50
  }
]
```

Fonte: Elaborada pelo autor

3.1.5 SPA

SPA significa Single Page Application, ou seja, uma aplicação de página única.

Uma SPA é uma aplicação web que funciona toda em uma só página, não necessitando de redirecionamento para outras páginas. O objetivo deste estilo de aplicação é proporcionar uma experiência de uso para o usuário semelhante a um aplicativo desktop (WIKIPEDIA, 2016)

Todo o código necessário para o funcionamento da página é carregado de uma só

vez, tendo o seu conteúdo carregado dinamicamente com o uso de AJAX ou de framework como AngularJS.

3.1.6 AngularJS

O AngularJS é um framework Javascript desenvolvido pela Google, usado para desenvolvimento de Front-End que permite a atualização dos dados em memória para elementos DOM e vice-versa, sem precisar realizar programação direta sobre a árvore DOM (*Two-Way Data Binding* ou *Ligação de Dados Bidirecional*) (ANGULARJS, 2010c). Além desse recurso, proporciona diversas funcionalidades, como XMLHttpRequest, application routing promises, através do uso de serviços modulares que podem ser injetados na aplicação.

Hoje é um dos frameworks mais utilizados em sistemas web (CLASSEN, 2015). Segundo pesquisa realizada no serviço Google Trends, estava isolado na quantidade de buscas dentre os quatro maiores frameworks Javascript para Front-End em 2015 (LUPCHINSKI, 2015).

Basicamente, neste framework o programador implementa controladores, os quais manipulam as informações advindas dos modelos de dados definidos no escopo da aplicação. Esses modelos são simples objetos Javascript que podem ser acessados também a partir dos elementos HTML com o uso da diretiva ng-bind ou duplas chaves (ANGULARJS, 2010b). Qualquer alteração dos dados dos modelos é exibida automaticamente na página (ANGULARJS, 2010a).

O AngularJS permite a definição de URLs internas as páginas da aplicação. Este recurso chamado de *application routing* é essencial para construir aplicações de página única pois permite que sejam especificadas URLs para exibição de conteúdo dentro elementos da página marcados com a diretiva ng-view.

Outro recurso muito importante são as *promises*. Uma promises basicamente é um objeto que controla o retorno de um função assíncrona (SCHLUETER, 2014). Muitas vezes necessitamos usar funções assíncronas e estas funções são executadas em um fluxo diferente do fluxo principal da aplicação, por isso não podemos esperar que o retorno dos dados da função esteja disponível na execução da próxima instrução no código.

Por exemplo, em uma consulta aos dados de uma página (que custa tempo para ser feita), necessitamos tomar ações após a chegada do resultado ou em uma situação de erro inesperada. Essas ações são programadas em funções chamadas de funções de *callback* e

são invocadas quando uma operação assíncrona é concluída ou gera algum erro.

3.1.7 Angular Material

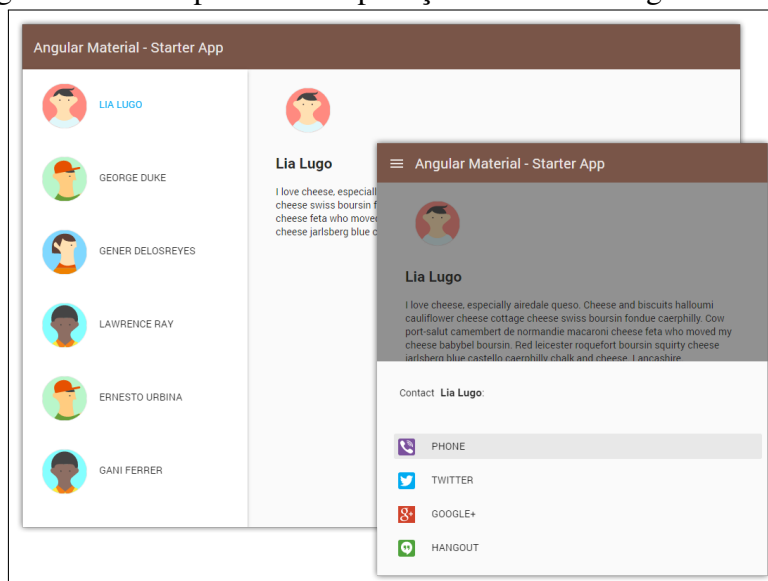
O Angular Material é um framework desenvolvido por uma comunidade de designers e programadores, que implementa as especificações do Material Design da Google. É um projeto com componentes de interface reutilizáveis, com acessibilidade baseados no Material Design (ANGULAR-MATERIAL, 2016).

O Material Design tem por objetivo criar uma linguagem visual que sintetize os princípios de boas práticas de design, permitindo um sistema proporcione uma experiência unificada entre as plataformas e tamanhos de dispositivos. Os preceitos das aplicações móveis são fundamentais, mas o toque, voz, mouse e teclado também são importantes (MATERIAL-DESIGN, 2016).

Este framework facilita o desenvolvimento da interface pelo programador sem conhecimentos avançados de UI, pois os seus componentes são facilmente manipulados através de tags próprias com atributos autoexplicativos. O estilo e as cores dos componentes são gerenciáveis por configuração, ainda que às vezes seja necessário ajustes manuais para obtermos os resultados desejados.

Na figura 3.2 tem-se um exemplo de uma aplicação construída usando Angular Material. Pode-se observar que o estilo de interface da aplicação tem semelhança com as aplicações nativas Android feitas pela empresa Google, mantenedora desse projeto.

Figura 3.2: Exemplo de uma aplicação utilizando Angular Material



Fonte: <https://github.com/angular/material-start/>

3.2 Back-End

Nesta seção são apresentadas as tecnologias e conceitos ligados ao Back-end deste projeto.

No Back-end de uma aplicação web estão os componentes do sistema que recebem os dados informados pelo usuário na camada de Front-end. Esses dados são processados no Back-end, podendo ser armazenados em bancos de dados e responder ao Front-end com novas informações.

O Back-end deste projeto têm as seguintes tecnologias funcionando em conjunto: Node.js, Express e MongoDB. Eles são descritos a seguir.

Na arquitetura definida neste trabalho o Back-end está encarregado de fazer validações de dados, autenticação, persistência em bancos de dados, redirecionamento das páginas e o fornecimento geral dos serviços REST.

3.2.1 Node.js

O Node.js é um ambiente de execução de Javascript construído sobre a engine V8 usada no navegador Chrome. É um ambiente de execução orientado a eventos assíncronos, desenvolvido com o foco de criar aplicações de rede altamente escaláveis (NODE.JS, 2016).

A ideia da arquitetura do Node.js é utilizar comunicação de entrada e saída não bloqueante e processamento assíncrono para obter menor tempo de resposta às requisições quando há muitas conexões concorrendo simultaneamente (STRONG-LOOP, 2014).

Neste projeto, o Node.js é a base onde o framework Express será executado e onde serão realizadas as regras de negócio mais sensíveis do sistema, a autenticação e comunicação com a base de dados.

3.2.2 Sistema de pacotes NPM

O NPM é um sistema de gerenciamento de pacotes que permite a distribuição de projetos, de forma pública ou privada entre equipes, permitindo o reuso de código com maior facilidade.

A ideia principal do NPM é criar pequenos pacotes que combinados a outros paco-

tes permitam a criação de projetos maiores, fazendo com que equipes focadas em problemas específicos possam beneficiar equipes com problemas mais genéricos (NPM, 2016b).

Basicamente, um pacote é composto de um diretório com um ou mais arquivos e diretórios que também possui um arquivo com o nome “package.json” com metadados sobre o pacote (NPM, 2016a).

3.2.3 REST

Representational state transfer (REST) é uma forma de prover interoperabilidade na internet usando de URLs para fornecer serviços sobre recursos dos sistemas.

O REST estabelece algumas restrições arquiteturais para definir um sistema RESTful, de forma a garantir performance, escalabilidade, simplicidade, modificabilidade, visibilidade, portabilidade e confiabilidade (FIELDING, 2000). Neste trabalho não é garantido o cumprimento de toda a especificação REST, mas sim algumas características de portabilidade e interoperabilidade que são interessantes para a organização do sistema.

Basicamente um sistema que implementa um acesso aos seus dados pelo conceito REST utiliza URLs onde fornece operações sobre um dado recurso.

Por exemplo, se alguém deseja disponibilizar dados dos usuários, pode fornecer a URL <http://www.exemplo.org/usuarios/> para que seja possível realizar operações diversas sobre usuários usando verbos HTTP, como segue:

- **GET:** Lê um recurso específico, ou uma coleção de recursos. Um acesso à um usuário com um identificador 1234, seria feito usando a URL <exemplo.org/usuarios/1234> com o método GET.
- **PUT:** Atualiza um recurso específico, ou uma coleção de recursos. Uma atualização sobre o usuário 1234 se daria pela URL <exemplo.org/usuarios/1234> enviando os dados necessários para a atualização pelo método PUT.
- **DELETE:** Remove um recurso específico pelo identificador. A remoção do usuário 1234 pode ser feita usando o método DELETE sobre a URL <exemplo.org/usuarios/1234>.
- **POST:** Cria um novo recurso. A criação de um novo usuário é feita encaminhando os dados deste para a URL <exemplo.org> usando o método POST.

3.2.4 Express

Express é um framework escrito em Javascript que roda no ambiente Node.JS, focado na simplicidade, facilitando a criação de APIs baseadas em REST. Este framework trabalha com o conceito de roteadores, que são funções programadas para responder às requisições HTTP conforme o método utilizado (HAHN, 2016).

Um roteador responde em um ou mais endereços URL, podendo responder e realizar ações diferenciadas a cada diferente método de requisição (HTTP verbs). Neste projeto, o Express faz o direcionamento das páginas e fornece a API de estilo REST da qual nosso Front-end irá utilizar nas requisições de dados do sistema.

O Express ainda fornece os middlewares, que são funções que recebem uma requisição, fazem alguma ação sobre esta e repassam para o próximo middleware ou para o roteador de destino. Os middlewares são usados para implementação de funções como autenticação, validação, log, entre outras.

3.2.5 Passport

Passport nada mais é do que um Middleware para a plataforma Node/Express. Serve especificamente para a tarefa de autenticação de requisições. Este módulo facilita o desenvolvimento, oferecendo diversas estratégias de autenticação, provendo desde autenticações mais básicas utilizando usuário e senha, até autenticações integradas com as redes sociais mais populares (PASSPORT, 2016).

Este projeto usa uma estratégia simples, realizando a autenticação pelas informações de usuário e senha, porém caso seja necessário, pode-se facilmente utilizar outras formas de validação de acesso.

3.2.6 MongoDB

MongoDB é um banco de dados open-source baseado em documentos em formato similar ao JSON apresentado na subseção 3.1.4.

Um registro no banco MongoDB é um documento estruturado em pares de campos e valores. Os valores podem ser outros documentos, arrays ou arrays de documentos.

Um conjunto de documentos fica armazenado em uma coleção. Sendo que estes

documentos não possuem um esquema fixo de campos e tipos de dados, diferentemente dos sistemas de gerenciamento de banco de dados SQL clássicos.

As vantagens atribuídas a este modelo de banco de dados segundo o fabricante (MONGODB, 2016) são:

- Documentos correspondem a tipos de dados nativos em muitas linguagens de programação.
- Documentos embutidos e arrays reduzem a necessidade do uso de joins diminuindo o custo de processamento.
- Esquemas dinâmicos suportam um polimorfismo fluente (documentos da mesma coleção podem ter campos diferentes).

Neste projeto, a maior vantagem obtida ao utilizar este banco de dados foi a facilidade de realizar a interpretação dos dados em formato *JSON*, já que foi usado *Javascript* como linguagem principal do projeto.

A justificativa para a escolha deste banco de dados neste projeto é a possibilidade recuperar os dados da base diretamente no formato *JSON* sem a necessidade de qualquer transformação dos dados. Dessa forma obtém-se uma uniformidade no formato dos dados, tanto na base de dados quanto no back-end e front-end pois todos utilizam *JSON* como formato de dados.

3.2.7 Mongoose

O Mongoose é um framework para Node.JS utilizado para facilitar a validação, casting de dados e lógica de negócio, fornecendo uma solução baseada em esquemas para modelagem dos dados da aplicação.

Neste framework é possível definir os campos das entidades do banco de dados através de objetos chamados de Schema. Os Schemas que permitem que programador possa agregar validação dos tipos de dados inseridos na base de dados, sem a necessidade de programação manual (MONGOOSE, 2016c).

Todas as operações de criação, recuperação, alteração e remoção podem ser feitas através de funções fornecidas por objetos chamados Model que são compilados sobre os Schemas (MONGOOSE, 2016a).

O uso deste framework possibilita maior facilidade ao programador, já que não necessitará saber da sintaxe específica do banco de dados utilizado, e sim apenas os co-

mandos utilizados pelo Mongoose. Além disso, permite ao sistema desacoplar as suas regras de negócio do banco de dados utilizado já que o framework abstrai a linguagem de consulta nativa da base de dados.

4 PROJETO DA APLICAÇÃO

Neste capítulo é apresentado o projeto do novo sistema AvalWeb[®].

Na seção 4.1 apresenta-se as funcionalidades desejadas na forma de *user stories*, para os atores do sistema denominados professor e aluno.

Na seção 4.2 modela-se as entidades que irão compor a base de dados do sistema.

Na seção 4.3 descreve-se a *API REST* construída para interação com a camada Front-end do sistema.

Na seção 4.4 mostra-se como está organizada a navegação no sistema utilizando um diagrama de atividades.

4.1 User Stories

Baseado nos trabalhos anteriores do sistema *AvalWeb*[®] e ideias mais recentes, foram definidas *user stories* ou *histórias de usuário*, para estabelecer os requisitos do sistema.

Em uma *user story* são descritas funcionalidades de valor para o usuário de um sistema de software (COHN, 2004). Geralmente utiliza-se um modelo de frase para descrever uma *user story*, (PRIMO, 2011) como este que segue:

“Como um <tipo do usuário>, eu quero <objetivo> para que <alguma razão>.”

Por exemplo, um atendente de uma biblioteca precisa buscar livros cadastrados no sistema para fazer reserva para um estudante. Neste caso pode-se escrever esta funcionalidade na forma de história de usuário da seguinte forma:

“Como atendente da biblioteca, quero pesquisar exemplares de livros para que eu possa fazer reservas para os alunos.”

Nas subseções seguintes serão descritas as histórias de usuário dos dois atores do sistema: professor e aluno.

4.1.1 Stories do professor

As histórias de usuário do professor focam nas atribuições de um professor no contexto da avaliação online ou em sala de aula, e em atividades que o aluno não poderia ter acesso devido à questões de prevenção de fraudes.

Segue as histórias que serviram de base para o professor no sistema:

- *Como professor quero aplicar avaliações às minhas turmas com facilidade para que possa acompanhar com frequência a aprendizagem delas.*
- *Como professor quero aplicar avaliações escolhendo apenas os assuntos que gostaria de avaliar na prova para que seja mais rápido elaborar testes.*
- *Como professor quero acessar o sistema com segurança, para que ninguém não autorizado possa obter as respostas das minhas provas.*
- *Como professor quero cadastrar questões para manter um banco de questões.*
- *Como professor quero que minhas questões não sejam acessíveis a todos no sistema, para que não haja vazamento de questões.*
- *Como professor quero classificar uma questão pelo seu nível de dificuldade para que eu possa avaliar de forma justa os alunos.*
- *Como professor quero cadastrar categorias para classificar para as questões.*
- *Como professor quero estabelecer a hierarquia entre as categorias de questões para organizá-las.*
- *Como professor quero cadastrar alunos de forma rápida através de convites por e-mail para minhas turmas.*
- *Como professor quero verificar o desempenho individual dos meus alunos para que possa medir a aprendizagem deles.*
- *Como professor quero aplicar avaliações às minhas turmas para que possa acompanhar a aprendizagem delas.*
- *Como professor quero saber quais as questões que meus alunos mais erram para que eu possa reforçar o conteúdo nestes assuntos.*

4.1.2 Stories do aluno

As histórias de alunos descrevem atribuições do aluno no sistema, que são bem menores que as dos professores, mas não menos importantes.

- *Como aluno quero listar minhas avaliações no sistema para que eu possa respondê-las.*
- *Como aluno quero ver meu desempenho ao fim de uma prova para possa ver as áreas onde tenho dificuldades.*

- *Como aluno quero atualizar meus dados cadastrais, para que possa manter atualizadas minhas informações.*
- *Como aluno quero aprender durante uma avaliação caso erre para que possa melhorar meu desempenho em outras avaliações.*

4.2 Entidades

Nesta seção é detalhada cada entidade que compõe o sistema, descrevendo os campos e as funcionalidades de cada coleção da base de dados.

Apesar da base de dados MongoDB ser flexível quanto aos tipos de dados e campos armazenados em cada objeto de uma coleção, aqui documentamos os campos e tipos de dados utilizados pelas coleções, para que possamos ter uma melhor compreensão sobre o funcionamento do sistema.

Ao final da seção apresenta-se um modelo gráfico que representa as entidades do sistema e como elas interagem.

Como não há estudos sistemáticos sobre a melhor forma de representar graficamente a estrutura de uma base de dados baseada em documentos (VERA, 2015), utiliza-se neste trabalho a notação UML para diagrama de classes com algumas restrições, de forma a ilustrar globalmente a modelagem para que tenhamos um melhor entendimento do sistema.

4.2.1 Questions

Esta entidade representa uma questão do banco de questões. Armazena o enunciado da questão, a ilustração que pode acompanhá-la, as alternativas de resposta para a questão, a dificuldade da questão, categoria e o dono da questão. Segue a descrição dos abaixo:

- **_id**: Identificador único da questão no sistema, gerado automaticamente pelo banco de dados, quando da criação de um novo documento.
- **imagePath**: String que armazena o caminho onde está a imagem que ilustra a questão. Este campo pode estar preenchido, ou não, caso a questão não use imagem ilustrativa.
- **answers**: É um array de objetos que contêm informações sobre as respostas. Cada

resposta contém atributos que definem a sua formação, como segue:

answer: É uma String que guarda a resposta escrita.

feedback: É uma String que armazena um feedback para esta resposta.

rightAnswer: É um valor booleano que informa que a resposta é a certa caso seja verdadeiro.

- **difficulty:** Valor inteiro (1 a 3) que armazena a dificuldade da questão.
- **category:** String que armazena um ObjectID indicando que a questão está em uma categoria de questões.
- **owner:** String que indica o usuário dono da questão. Somente o dono da questão pode editar ou remover esta questão.

4.2.2 Categories

Esta entidade representa uma categoria de questões. Armazena o nome da categoria, as questões vinculadas a esta categoria, as subcategorias abaixo desta, a categoria pai e o usuário dono desta categoria. Segue a descrição dos campos abaixo:

- **_id:** Identificador único da categoria, gerado automaticamente pelo banco de dados, quando da criação de um novo documento.
- **name:** String que armazena o nome da categoria.
- **questions:** É um array de referências aos documentos da coleção Question.
- **subCategories:** É um array de referências aos documentos da coleção Categories, indicando as subcategorias imediatamente inferiores uma categoria.
- **superCategory:** É uma String que armazena um ObjectID indicando uma Category que contém a categoria em questão. Se o campo estiver com uma String vazia então a categoria não tem pai, sendo uma categoria raiz.
- **owner:** String que indica o usuário dono da categoria. Somente o dono da categoria pode visualizar, editar ou remover esta categoria.

4.2.3 Students

Esta entidade representa um estudante. Armazena o nome, data de nascimento, uma foto e o e-mail do aluno. Segue a descrição dos campos abaixo:

- **_id**: Identificador único do aluno, gerado automaticamente pelo banco de dados, quando da criação de um novo documento.
- **name**: String que armazena o nome do aluno.
- **birthDate**: Objeto Date que armazena a data de nascimento do aluno.
- **photo**: String que armazena o caminho de um arquivo de imagem que contém a foto do aluno.
- **email**: String que contém o e-mail do aluno.

4.2.4 Classes

Esta entidade representa uma turma de estudantes. Armazena o nome que identifica a turma, os estudantes que pertencem a ela e o usuário que criou a turma sendo o dono desta. Segue a descrição dos campos abaixo:

- **_id**: Identificador único da turma, gerado automaticamente pelo banco de dados, quando da criação de um novo documento.
- **name**: String que armazena o nome da turma.
- **students**: É um array de referências aos documentos da coleção Students, indicando os alunos que compõem esta turma.
- **owner**: String que indica o usuário dono desta turma, somente o dono da turma consegue visualizar, editar ou remover esta turma.

4.2.5 Users

Esta entidade representa um usuário do sistema. Armazena o nome de usuário, o tipo do usuário e a senha de acesso. Segue a descrição dos campos abaixo:

- **_id**: Identificador único de usuário, gerado automaticamente pelo banco de dados, quando da criação de um novo documento.
- **name**: String que armazena o nome da turma.
- **students**: É um array de referências aos documentos da coleção Students, indicando os alunos que compõem esta turma.
- **owner**: String que indica o usuário dono desta turma, somente o dono da turma consegue visualizar, editar ou remover seus dados.

4.2.6 Tests

Esta entidade representa uma avaliação cadastrada no sistema. Armazena o nome do teste, a data de aplicação, as questões ou categorias que a compõem, as turmas em que a avaliação será aplicada, a estratégia de aplicação, o tipo de avaliação e dono da avaliação. Segue a descrição dos campos abaixo:

- **_id**: Identificador único da avaliação, gerado automaticamente pelo banco de dados, quando da criação de um novo documento.
- **name**: String que armazena o nome da avaliação.
- **questions**: É um array de referências aos documentos da coleção Questions, indicando as questões que estão nesta avaliação.
- **categories**: É um array de referências aos documentos da coleção Categories, indicando as categorias que estão nesta avaliação.
- **classes**: É um array de referências aos documentos da coleção Classes, indicando as turmas que responderão a esta avaliação.
- **strategy**: String que indica estratégia de aplicação para esta avaliação. Atualmente no sistema temos duas opções: “fixed_order” representando a prova com ordem fixa de apresentação das questões e “dif_balance”, sendo a estratégia que apresenta uma questão mais fácil sobre o assunto atual, toda vez que o aluno erra alguma questão difícil sobre o assunto atual.
- **type**: Número inteiro que indica o tipo de prova sendo 1 para prova por categorias e 2 para prova por questões.
- **owner**: String que indica o usuário dono desta avaliação. Somente o dono da avaliação consegue editar ou remover seus dados.

4.2.7 Student Tests

Esta entidade representa uma avaliação realizada por um aluno no sistema. Esta estrutura armazena informações sobre a realização da avaliação e também informações de estado da execução da avaliação no lado servidor evitando a manipulação dos dados por parte do cliente. Segue a descrição dos campos abaixo:

- **_id**: Identificador único da avaliação, gerado automaticamente pelo banco de dados,

quando da criação de um novo documento.

- **user**: String que armazena o usuário que está sendo avaliado.
- **student**: Referência ao um documento da coleção Students, que representa o estudante que está fazendo a avaliação.
- **test**: Referência ao um documento da coleção Tests, que representa a avaliação sendo realizada.
- **date**: Armazena as informações de data e hora em que a avaliação começou.
- **finished**: Valor booleano que é verdadeiro quando a prova foi feita até fim e falso caso contrário.
- **answeredQuestions**: É um array de referências aos documentos da coleção Questions, indicando as questões que foram respondidas até o momento na avaliação.
- **answeredCategories**: É um array de referências aos documentos da coleção Categories, indicando as categorias que foram respondidas até o momento na avaliação.
- **answers**: É um array de referências às respostas selecionadas pelo aluno a cada questão.
- **answersGrade**: É um array de valores numéricos que define a nota da resposta sobre as questões respondidas. Os valores estão entre 0 e 1, sendo 0 a questão totalmente errada e 1 a questão totalmente certa.
- **answeredCategories**: É um array de referências aos documentos da coleção Categories, indicando as categorias que deverão ser respondidas até o fim da avaliação.
- **repeatedCategory**: É um array de valores booleanos que é usado quando a avaliação usa categorias. Marca quais categorias que estão tendo questões repetidas. Usado como informação para estratégias de aplicação.
- **actualCategory**: Número inteiro que armazena o índice atual do array de categorias durante a prova. Utilizado para saber qual a próxima categoria.
- **numberRetries**: Número que armazena a quantidade de vezes que o usuário errou uma questão. Informação utilizada pelas estratégias de aplicação.

4.2.8 Diagrama de entidades

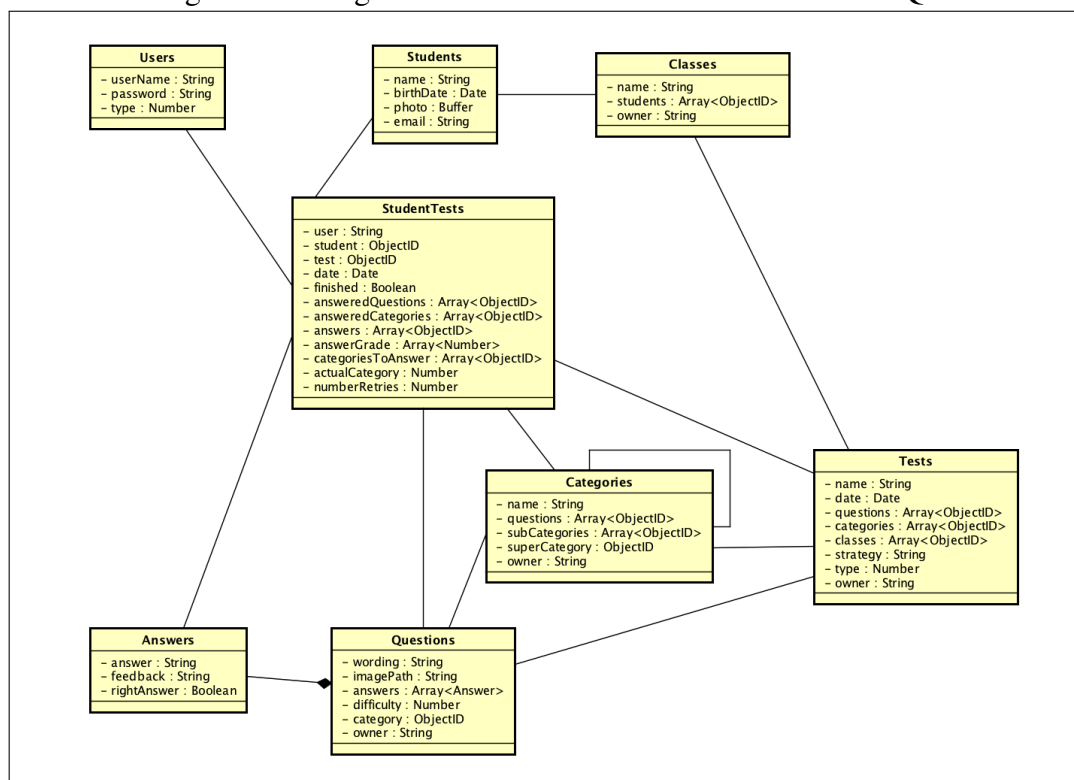
Nesta seção descreve-se como as entidades estão interligadas utilizando um diagrama de classes UML.

Para isto, estabeleceu-se que as associações consideradas mais importantes serão explicitadas. Enquanto as menos importantes terão apenas os seus campos informados dentro das classes. No caso de documentos embutidos utilizou-se a notação de associação de composição.

Na figura 4.1 pode-se observar como o sistema atualmente tem suas entidades associadas. A entidade *StudentTests* tem associações com quase todas entidades do sistema.

Isso porque ela contém as informações para a regra de negócio que envolve o fornecimento das questões ou categorias para realização da avaliação e ainda mantém as informações de estado de execução da avaliação.

Figura 4.1: Diagrama das entidades da base de dados NoSQL



Fonte: Elaborada pelo autor

4.3 API REST

Esta seção explica a API REST desenvolvida com o intuito de dar interoperabilidade ao sistema descrevendo cada funcionalidade com sua URL, métodos, parâmetros e retornos. Nas subseções a seguir apresenta-se as funcionalidades na forma de tabelas.

4.3.1 Questions

Tabela 4.1: Tabela de comandos para o recurso questions

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id da questão	Recuperar uma questão da base de dados	Objeto em formato JSON
/search/	GET	:filter - texto de busca	Filtra por questões com ocorrências do filtro	Array de objetos questions
/upload/	POST	:files - nomes de arquivos	Envia arquivos para a questão	Nome dos arquivos enviados
/	POST	JSON no formato da entidade questions	Inserir uma nova questão na base de dados	String 'question saved' em caso de sucesso
/id/:id	PUT	:id - id da questão e JSON no formato da entidade questions	Altera uma questão na base de dados	String 'question saved' em caso de sucesso
/id/:id	DELETE	:id - id da questão	Remove a questão conforme o id parâmetro	String 'question removed' em caso de sucesso

4.3.2 Categories

Tabela 4.2: Tabela de comandos para o recurso categories

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id da categoria	Recuperar uma categoria da base de dados	Objeto em formato JSON
/search/	GET	:filter - texto de busca	Filtra por categorias com ocorrências do filtro	Array de objetos category
/	POST	JSON no formato da entidade categories	Insere uma nova categoria na base de dados	String 'category saved' em caso de sucesso
/id/:id	PUT	:id - id da categoria e JSON no formato da entidade categories	Altera uma categoria na base de dados	String 'category saved' em caso de sucesso
/id/:id	DELETE	:id - id da categoria	Remove a categoria conforme o id parâmetro	String 'category removed' em caso de sucesso

4.3.3 Students

Tabela 4.3: Tabela de comandos para o recurso students

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id do aluno	Recuperar um aluno da base de dados	Objeto em formato JSON
/search/	GET	:filter - texto de busca	Filtra por alunos com ocorrências do filtro	Array de objetos student
/	POST	JSON no formato da entidade students	Insere um novo aluno na base de dados	String 'student saved' em caso de sucesso
/id/:id	PUT	:id - id do aluno e JSON no formato da entidade students	Altera um aluno na base de dados	String 'student saved' em caso de sucesso
/id/:id	DELETE	:id - id do aluno	Remove o aluno conforme o id parâmetro	String 'student removed' em caso de sucesso

4.3.4 Classes

Tabela 4.4: Tabela de comandos para o recurso classes

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id da turma	Recuperar uma turma da base de dados	Objeto em formato JSON
/search/	GET	:filter - texto de busca	Filtra por turmas com ocorrências do filtro	Array de objetos class
/	POST	JSON no formato da entidade classes	Inserir uma nova turma na base de dados	String 'class saved' em caso de sucesso
/id/:id	PUT	:id - id da turma e JSON no formato da entidade classes	Altera uma turma na base de dados	String 'class saved' em caso de sucesso
/id/:id	DELETE	:id - id da turma	Remove a turma conforme o id parâmetro	String 'class removed' em caso de sucesso

4.3.5 Users

Tabela 4.5: Tabela de comandos para o recurso users

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id do usuário	Recuperar um usuário da base de dados	Objeto em formato JSON
/search/	GET	:filter - texto de busca	Filtra por usuários com ocorrências do filtro	Array de objetos user
/	POST	JSON no formato da entidade users	Inserir um novo usuário na base de dados	String 'user saved' em caso de sucesso
/id/:id	PUT	:id - id do usuário e JSON no formato da entidade users	Altera um usuário na base de dados	String 'user saved' em caso de sucesso
/id/:id	DELETE	:id - id do usuário	Remove o usuário conforme o id parâmetro	String 'user removed' em caso de sucesso

4.3.6 Tests

Tabela 4.6: Tabela de comandos para o recurso tests

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id da avaliação	Recuperar uma avaliação da base de dados	Objeto em formato JSON
/search/	GET	:filter - texto de busca	Filtra por avaliações com ocorrências do filtro	Array de objetos test
/	POST	JSON no formato da entidade tests	Insere um novo usuário na base de dados	String 'test saved' em caso de sucesso
/id/:id	PUT	:id - id da avaliação e JSON no formato da entidade tests	Altera uma avaliação na base de dados	String 'test saved' em caso de sucesso
/id/:id	DELETE	:id - id da avaliação	Remove a avaliação conforme o id parâmetro	String 'test removed' em caso de sucesso

4.3.7 Student Tests

Tabela 4.7: Tabela de comandos para o recurso studenttests

URL	Método	Parâmetros	Funcionalidade	Retorno
/id/:id/	GET	:id - id da avaliação feita pelo aluno	Recuperar a avaliação do aluno da base de dados	Objeto em formato JSON
/finishedtest/:id	GET	:id - id da avaliação feita pelo aluno	Recuperar avaliação do aluno caso já tenha sido feita	Objeto em formato JSON
/search/	GET	Sem parâmetros	Retorna as avaliações associadas ao aluno conectado	Array com as avaliações aplicadas nas turmas do aluno conectado
/starttest/	POST	JSON no formato da entidade tests	Faz o aluno conectado iniciar uma avaliação	Retorna a primeira questão para o aluno responder
/checkquestion/	POST	JSON no formato da entidade questions	Verifica se o aluno acertou a questão	Retorna questão em formato JSON com campo booleano "right" indicando se o aluno acertou a questão
/nextquestion/	POST	JSON no formato da entidade questions	Seleciona uma nova questão para o aluno	Retorna questão em formato JSON

4.3.8 Invites

Tabela 4.8: Tabela de comandos para o recurso invites

URL	Método	Parâmetros	Funcionalidade	Retorno
/	POST	Array com objetos JSON contendo os campos email e name	Convida por e-mail alunos para acesso ao sistema fornecendo uma senha.	String 'emails sent' em caso de sucesso

4.3.9 Reports

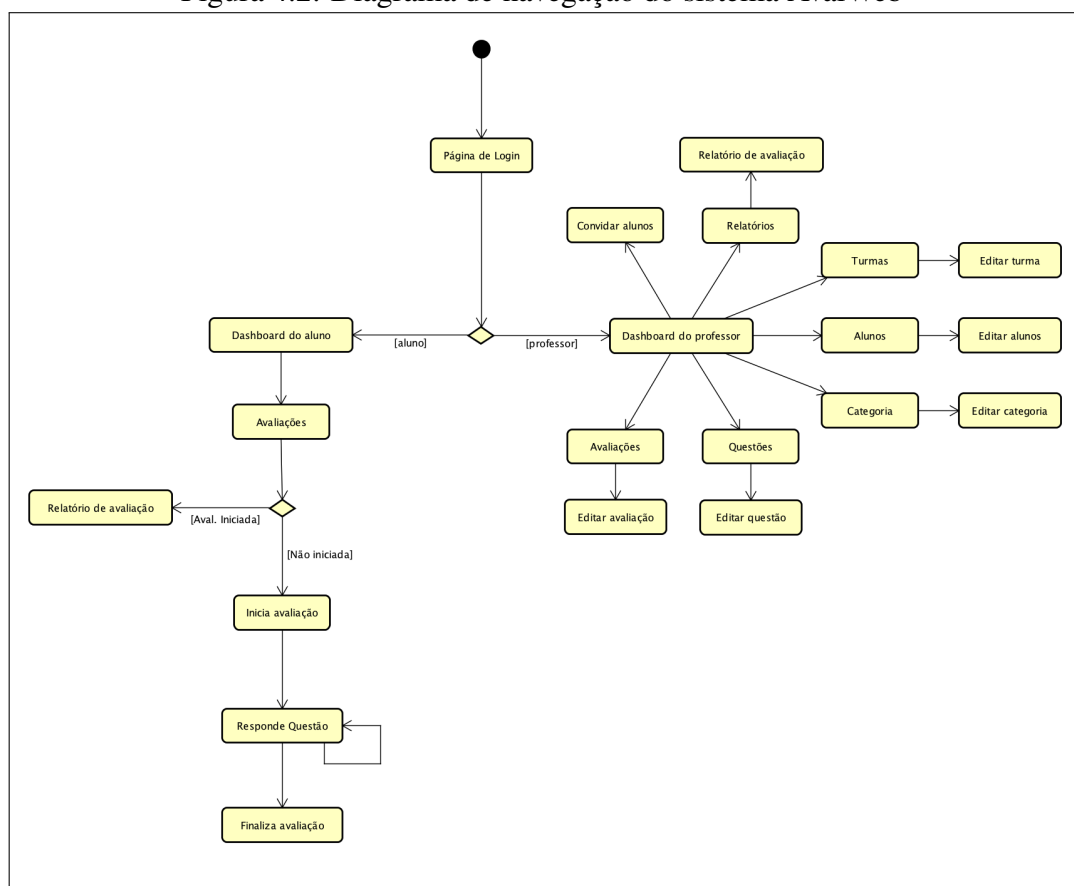
Tabela 4.9: Tabela de comandos para o recurso reports

URL	Método	Parâmetros	Funcionalidade	Retorno
/studentswithscore/:id	GET	:id - id do aluno	Calcula a nota dos testes realizados pelo aluno	Array com as avaliações e as notas obtidas pelo aluno

4.4 Diagrama de navegação

Nesta seção apresenta-se um diagrama de navegação com objetivo de mapear as principais interfaces implementadas no sistema. Para isso, usou-se um diagrama de atividades UML, o que possibilita uma visualização de fácil entendimento dos fluxos de trabalho dentro do sistema.

Figura 4.2: Diagrama de navegação do sistema AvalWeb[®]



Fonte: Elaborada pelo autor

Na figura 4.2 pode-se observar que há atividades separadas em dois grandes grupos: as atividades de professor e as atividades de aluno.

As atividades do professor são disponibilizadas a partir da dashboard do professor. As atividades do professor estão baseadas nas *user stories* descritas na seção 4.1.1, de forma análoga as atividades de aluno estão baseadas nas *user stories* descritas na seção 4.1.2.

5 IMPLEMENTAÇÃO DO PROJETO

Neste capítulo são apresentadas de forma mais prática as tecnologias utilizadas na execução do projeto, mostrando comandos, códigos, configurações e demais detalhes com o objetivo de demonstrar o funcionamento do sistema.

Na seção 5.1 é exibida a configuração do projeto utilizando o sistema de pacotes NPM.

Na seção 5.2 é demonstrado como a autenticação é feita no sistema utilizando a biblioteca Passport.

Na seção 5.3 são apresentados os componentes e estrutura de organização dos arquivos do sistema conforme as suas funcionalidades.

Na seção 5.4 mostra-se o padrão de projeto strategy aplicado ao problema da mudança de estratégia de aplicação de provas.

5.1 Configuração de projeto NPM

Para configuração do projeto usando o NPM, deve-se criar um arquivo com o nome *package.json* na pasta raiz do projeto. Este arquivo contém as informações do nome do projeto, versão, scripts de inicialização do sistema, dependência de pacotes, entre outras opções que permitem que o projeto seja facilmente compartilhado e instalado em outros ambientes de execução.

Na figura 5.1 mostra-se o arquivo *package.json* gerado para o projeto, onde constam as dependências com outros pacotes e as versões utilizadas desses pacotes.

Com este pacote configurado pode-se instalar o AvalWeb[®] em um ambiente Node.js de forma muito simples apenas digitando o seguinte comando no console a partir do diretório raiz do projeto: `npm install`.

Figura 5.1: Configuração de pacote NPM do sistema AvalWeb®

```
{
  "name": "avalweb",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "angular": "^1.5.7",
    "angular-animate": "^1.5.7",
    "angular-aria": "^1.5.7",
    "async": "^2.1.2",
    "body-parser": "~1.13.2",
    "cookie-parser": "~1.3.5",
    "debug": "~2.2.0",
    "express": "~4.13.1",
    "express-session": "^1.7.6",
    "jade": "~1.11.0",
    "mongoose": "^4.5.0",
    "morgan": "~1.6.1",
    "nodemailer": "^0.7.1",
    "serve-favicon": "~2.3.0"
  }
}
```

Fonte: Elaborada pelo autor

5.2 Autenticação

A autenticação do AvalWeb® utiliza atualmente a estratégia de autenticação chamada *local* da biblioteca Passport.

Para realizar a autenticação, o Passport é configurado passando os nomes dos campos que se referem às informações de usuário e senha sendo necessário definir uma função que irá processar a validação das autenticações.

Na figura 5.2 pode-se ver o código utilizado no protótipo do sistema.

Pode-se perceber que no código que a função de validação verifica se existe um usuário com a senha informada e retorna um objeto com os dados de acesso para o cliente.

O tipo do usuário encontrado na base de dados é verificado para que o objeto estabelecido na sessão ao cliente seja construído conforme os acessos do usuário.

Essa estratégia foi utilizada dentro do escopo desta aplicação, mas futuramente poderá ser substituída por outros métodos mais flexíveis e integrados com aplicações diversas, como por exemplo, as redes sociais.

Figura 5.2: Função de autenticação do sistema

```

var LocalStrategy = require('passport-local').Strategy;
passport.use(new LocalStrategy({usernameField: 'user',
                                passwordField: 'password'},

function(username, password, done) {
  userModel.findOne({userName: username, password: password})
  .exec(function(err, user){
    if (err) { return done(err); }
    if (!user) { return done(null, false); }
    if(user){
      if(user.type == 1){
        var u = {username: username, type: 'professor'};
        return done(null, u);
      }
      else{
        var u = {username: username, type: 'aluno'};
        return done(null, u);
      }
    }
  });
});
});

```

Fonte: Elaborada pelo autor

5.3 Organização dos componentes do Sistema

Nesta seção apresenta-se os componentes do sistema e como estes estão organizados na árvore de diretórios dentro do projeto.

5.3.1 Roteadores do Framework Express

Os roteadores do framework Express por padrão ficam localizados no diretório `routes` dentro do diretório raiz do projeto.

Esses roteadores respondem às requisições HTTP conforme a combinação da URL de destino e método utilizado, por isto definiu-se os roteadores com os parâmetros `HANDLER` e `URL` e `METHOD`.

Um `HANDLER` é uma função que processará a requisição encaminhada para a URL através de um método (`METHOD`) específico.

Como este projeto foi baseado em uma arquitetura *REST*, os roteadores são parte primordial do sistema, tendo a função de fazer toda e qualquer comunicação com a camada cliente.

Na figura 5.3 demonstra-se um exemplo de um código implementando o roteador que responderá a URL `student/id/` quando este receber uma requisição usando o método GET.

Os roteadores podem receber parâmetros de entrada a partir da URL, como no caso da figura 5.3, que recebe um parâmetro `:id` ou por envio de dados diretamente no objeto `req.body` da requisição.

Figura 5.3: Roteador simples de método GET

```
router.get('/id/:id', function(req, res, next){
  var query = studentModel.findById(req.params.id);
  query.exec(function(err, result){
    if (err)
      res.send(err);
    else{
      res.json(result);
    }
  });
});
```

Fonte: Elaborada pelo autor

5.3.2 Schemas do framework Mongoose

Os *schemas* do framework *Mongoose* permitem a definição de uma estrutura básica a qual é seguida neste projeto ao utilizar a base de dados *MongoDB* (MONGOOSE, 2016b). Isso garante a existência dos campos definidos no *schema* a cada novo documento criado, evitando erros quando tenta-se acessar campos inexistentes, já que o MongoDB permite que seus documentos não possuam estrutura fixa.

Esses *schemas* são transformados em objetos `Model` posteriormente e por isso neste projeto ficam localizados no diretório de mesmo nome.

Os objetos `Model` permitem inserir, alterar e remover novos documentos a partir de seus métodos de forma mais simples do que o acesso direto ao banco de dados.

Para ilustrar, segue um exemplo do *Schema* utilizado para modelar a entidade *Student* na figura 5.4. Percebe-se que são definidos os campos que se quer que o documento contenha no *Schema* e que na última linha é exportado o objeto `Model` baseado no *Schema*, para que os demais módulos do sistema possam acessar esta coleção também.

Figura 5.4: Schema da entidade Student e modelo gerado a partir do schema

```
var mongoose = require('mongoose');

var studentSchema = new mongoose.Schema({
  name : String,
  birthDate : Date,
  photo : String,
  email: String
});

module.exports = mongoose.model('Student', studentSchema);
```

Fonte: Elaborada pelo autor

5.3.3 Páginas estáticas com AngularJS

Neste projeto não é feito uso de qualquer processamento de renderização de páginas no servidor, assim remove-se este custo de processamento do lado servidor permitindo maiores velocidades de resposta. Por causa essa característica de projeto atribuí-se ao cliente uma maior responsabilidade sobre as questões relacionadas a interface do sistema.

O AngularJS possibilita a renderização de partes das páginas estáticas, tornando-as páginas dinâmicas através de controladores implementados em *Javascript* rodando no browser, assim obtém-se uma renderização de páginas que é processada pelo próprio navegador.

As páginas fornecidas aos clientes são construídas basicamente a partir de três arquivos na maioria dos casos, uma primeira página de listagem de dados, uma segunda página de detalhamento de dados e um arquivo Javascript que contém os `controllers` *AngularJS* que alimentam e atualizam os dados usados pelas páginas.

A primeira página é a entrada da funcionalidade, geralmente onde é exibida uma listagem de dados relacionados com a funcionalidade em questão. Os arquivos destas páginas tem o padrão de nomeação `<funcionalidade>List.html`, onde a tag `<funcionalidade>` geralmente se refere à entidade da qual estamos trabalhando nos dados.

Na segunda página é feito o detalhamento, alteração e inserção de dados. Estas páginas tem o formato de nome de arquivo `<funcionalidade>Dialog.html`, de forma análoga as páginas de listagem de dados, porém agora estas contém o formato da caixa de diálogo utilizada para o detalhamento.

Os arquivos Javascript de controle das páginas ficam armazenados no diretório `javascripts` localizado no diretório raiz do projeto. Geralmente são nomeados com o padrão de nome `<funcionalidade>Controller.js`. Estes arquivos contém toda

a lógica de renderização e atualização dos dados, fazendo a comunicação com o servidor por *XHR* e posteriormente renderizando os dados na página.

5.4 Padrão Strategy e estratégias de aplicação de avaliação

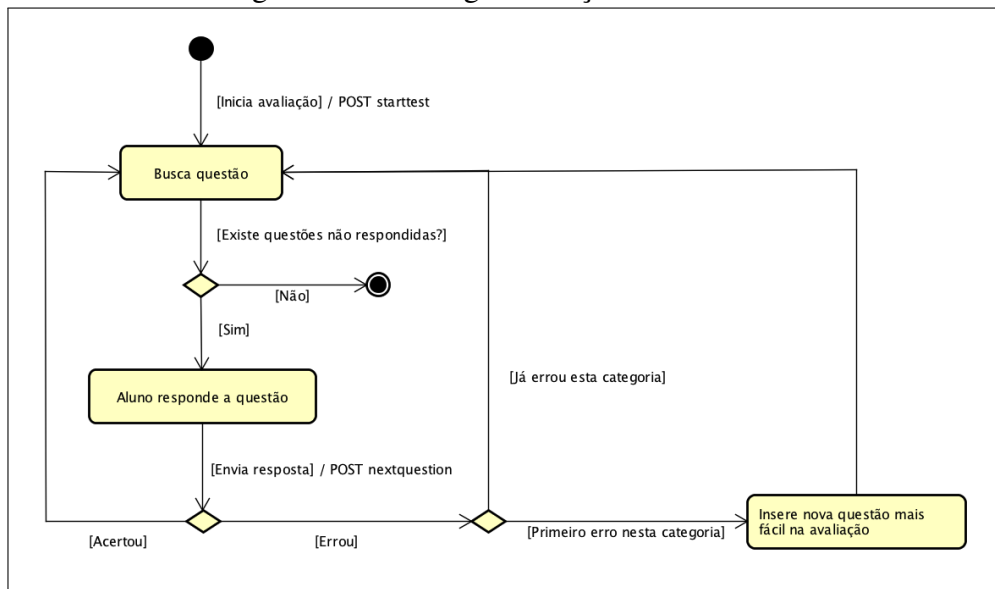
Para facilitar a criação de estratégias de aplicação de questões e categorias durante as provas, foi utilizado o padrão de projeto chamado *Strategy*, que permite a troca de algoritmos em tempo de execução.

Neste projeto foram implementadas duas estratégias de aplicação de avaliações até o momento.

A primeira estratégia utiliza a ordem das questões que está predefinida na criação da prova. Esta estratégia é chamada de *ordem fixa*.

A segunda estratégia é chamada de *balanço de dificuldade*. Ao cadastrar uma questão no sistema, o professor deverá classificar o seu nível de dificuldade. Esta informação é utilizada nesta estratégia da seguinte forma ilustrada na figura 5.5:

Figura 5.5: Estratégia balanço de dificuldade



Fonte: Elaborada pelo autor

Esta estratégia apresenta inicialmente questões cadastradas como difíceis. Durante a avaliação verificam-se as questões que foram respondidas de forma errada, apresentando no caso de um erro uma nova questão mais fácil desta mesma categoria.

A ideia desta estratégia é observar a profundidade do conhecimento do aluno sobre os assuntos abordados. Quando um aluno acerta questões difíceis assume-se que o aluno

compreende de forma aprofundada aquele assunto. Quando o aluno erra a questão, lhe é oferecida uma nova oportunidade para verificar se o aluno sabe ao menos questões básicas sobre o assunto.

Pode-se implementar outras estratégias com ideias novas utilizando extendendo a classe `TestStrategy` localizada no diretório raiz do projeto.

6 GUIA DE USO DO SISTEMA

Neste capítulo são apresentadas as funcionalidades implementadas no protótipo do sistema. As seções seguintes estão organizadas em interface do sistema, atividades do professor e as atividades do aluno. Nas subseções a seguir são detalhados os componentes visuais do sistema e suas funcionalidades e as ações permitidas a cada tipo de usuário em questão, explicando o fluxo de trabalho para realizar as atividades no sistema mostrando os resultados obtidos.

6.1 Interface do sistema

Esta seção explica os conceitos e funcionalidades aplicadas nas diversas telas do AvalWeb®. Ícones, botões, menus, e demais componentes de interface são descritos para melhor entendimento geral do uso do sistema.

6.1.1 Ícones de botões do sistema

Em várias atividades do AvalWeb® tem-se a padronização dos botões utilizando de ícones com símbolos que fazem analogias com as funcionalidades que estes executam, tornando o uso do sistema mais intuitivo. Nesta subseção descreve-se as atividades executadas por estes botões.

As atividades estão listadas no menu lateral do sistema. Este menu pode ser recolhido para obter-se uma maior visualização dos dados na tela. Para recolher o menu lateral deve-se pressionar o ícone indicado com uma borda vermelha na figura 6.1. No acesso via celular ou tablet podemos recolher o menu fazendo um movimento de “arrastar” em cima do menu, da direita para a esquerda. Ainda na figura 6.1 pode-se observar um indicador da atividade que está sendo executada atualmente, localizado ao lado do botão de recolhimento do menu. No caso exibido estamos na tela de atividades relacionadas ao cadastro de questões.

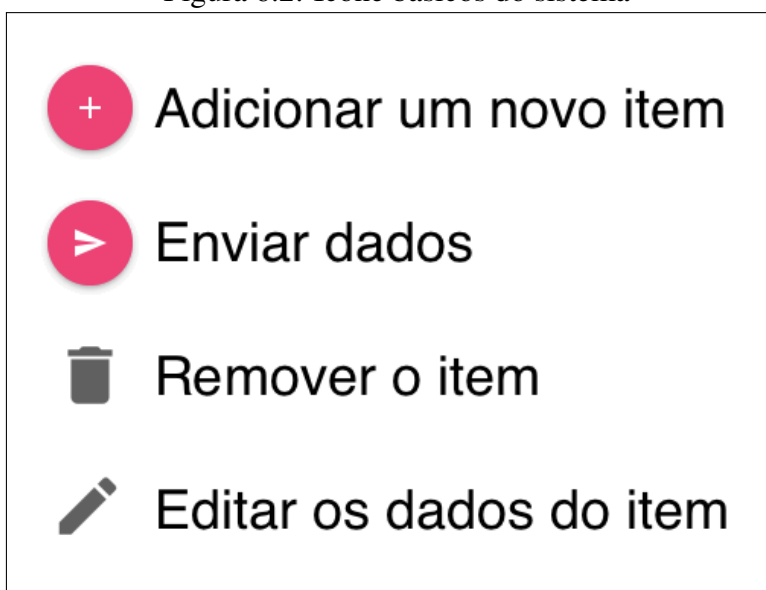
Figura 6.1: Botão de recolhimento de menu



Fonte: Elaborada pelo autor

O AvalWeb[®] faz uso de botões flutuantes localizados no canto inferior direito da tela e também dentro das listagens de dados a cada item da lista. Estes botões executam funções específicas vinculadas à tela que está sendo exibida, porém sempre têm o mesmo objetivo do ponto de vista dos dados que estão sendo modificados. Na figura 6.2 é mostrado uma listagem dos botões e ícones utilizados e a funcionalidade geral esperada de cada um deles.

Figura 6.2: Ícone básicos do sistema



Fonte: Elaborada pelo autor

6.2 Atividades do professor

Nesta seção são explicadas de forma prática através de exemplos e imagens as atividades do professor e o fluxo de trabalho para a criação de avaliações, questões, categorias e demais atividades relacionadas ao acesso de professor no sistema.

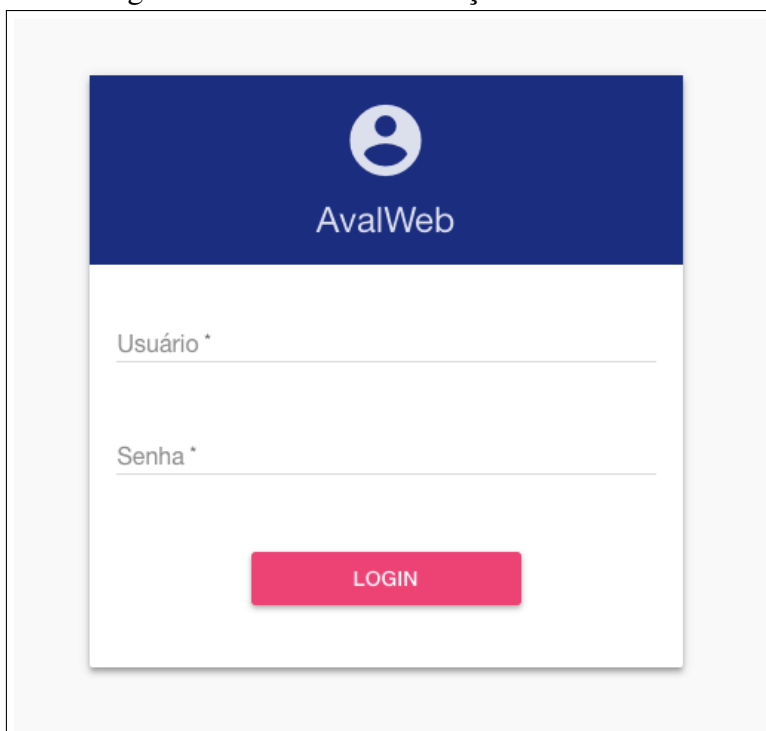
6.2.1 Autenticação

Para autenticação como professor no sistema é necessário obter cadastro de usuário específico, tal cadastro só pode ser realizado pelo administrador do sistema. Neste protótipo prova de conceito somente uma pessoa com acesso direto à base de dados pode criar um usuário de professor.

Com o login e senha cadastrados como professor no sistema em mãos, o professor deverá acessar o endereço onde está hospedado o AvalWeb®.

Será exibida uma tela de autenticação no AvalWeb® como demonstrado na figura 6.3. Após a inserção do usuário e da senha, o acesso poderá ser feito clicando no botão login ou apenas pressionando a tecla enter.

Figura 6.3: Tela de autenticação do AvalWeb®

A imagem mostra a interface de autenticação do sistema AvalWeb. No topo, há uma barra azul escura com um ícone de perfil de usuário em branco e o texto "AvalWeb" em branco. Abaixo, há um formulário branco com dois campos de entrada: "Usuário *" e "Senha *", cada um com uma linha de texto e uma linha de base. Abaixo dos campos, há um botão retangular de cor rosa com o texto "LOGIN" em branco.

Fonte: Elaborada pelo autor

6.2.2 Cadastro de questões

A tela de cadastro de questões possui as funções básicas de visualização, cadastro, edição e remoção de questões. Utilizando o botão geral de adicionar itens obtém-se a seguinte tela exibida na figura 6.4.

Figura 6.4: Tela de edição de questões



Fonte: Elaborada pelo autor

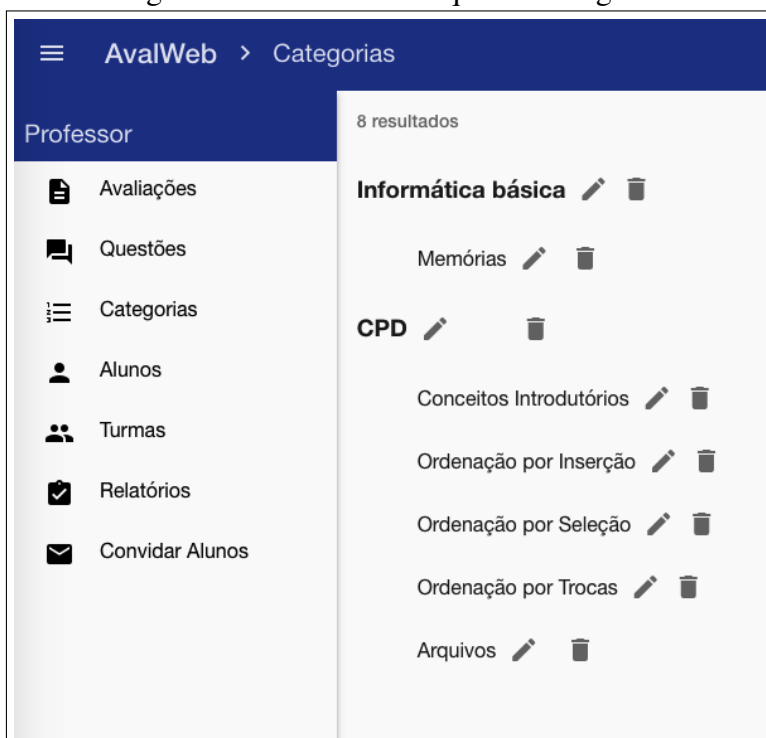
Nesta tela deve-se preencher os dados conforme indicados nos campos do formulário. Após o preenchimento pressiona-se no botão gravar para inserir a nova questão no banco de dados. Pode-se editar as questões cadastradas clicando em uma questão listada. Para remover uma questão, basta clicar no ícone que representa uma lixeira conforme foi apresentado na figura 6.2.

6.2.3 Cadastro de categorias

Na tela de cadastro de categorias é exibida a hierarquia das categorias cadastradas no sistema, como no exemplo da figura 6.5. Novamente como nas demais telas, o botão para adição de categorias é como o indicado na figura 6.2 assim como as opções de remoção e edição de categorias.

A tela de edição de categorias exhibe uma listagem das categorias para que possamos selecionar um categoria pai. Caso a categoria não possua pai, então deixamos este campo vazio.

Figura 6.5: Tela de hierarquia de categorias



Fonte: Elaborada pelo autor

6.2.4 Cadastro de alunos

A tela cadastro de alunos funciona da mesma forma que o cadastro de questões. Permite as operações de adição, edição e remoção de alunos a partir dos ícones padronizados do sistema.

6.2.5 Cadastro de turmas

No cadastro de turmas pode-se gerenciar as turmas criadas pelo professor. Uma turma deve ter os seus alunos selecionados na tela de criação de turmas. Para selecionar um aluno, deve-se clicar na chave de comutação localizada à direita de cada aluno listado. Para facilitar a localização de um aluno, é possível filtrar pelo nome do aluno utilizando o campo indicado com borda em vermelho na figura 6.6.

Figura 6.6: Tela de cadastro de turmas

Alunos	
João	<input checked="" type="checkbox"/> Matriculado?
João Alberto	<input checked="" type="checkbox"/> Matriculado?
José Luiz	<input type="checkbox"/> Matriculado?

CANCELAR GRAVAR

Fonte: Elaborada pelo autor

6.2.6 Convite de alunos

A tela de convite de alunos dá acesso aos alunos ao sistema AvalWeb[®]. Informando uma lista dos alunos em formato *CSV* com os seus respectivos nomes e e-mails, o sistema encaminhará um e-mail a cada aluno da lista, informando a senha gerada para acesso ao sistema.

Na figura 6.7, tem-se um exemplo de envio de convites aos alunos.

Figura 6.7: Convites por CSV

Lista de e-mails em formato CSV:

João Alberto, joãoalberto@email.com
 José da Silva Alberto, josedasilva@email.com
 Maria dos Santos, mariasantos@email.com

Senha para envio de emails

Exemplo:
 João da Silva,joaoasilva@email.com
 José de Souza,josedesouza@email.com

Fonte: Elaborada pelo autor

6.2.7 Cadastro de avaliações

O cadastro de avaliações é a atividade principal do AvalWeb® e por isso é mais complexa, pois depende do cadastro de diversos dados nas demais áreas do sistema. Para o cadastro de uma avaliação é recomendável ter cadastrado de antemão as categorias, questões, alunos e turmas, pois isso evitará que o usuário precise editar várias vezes a avaliação para inserir mais dados.

Na tela de avaliações tem-se uma listagem das provas cadastradas no sistema. Pode-se inserir, editar e remover avaliações usando os botões padrão do sistema para estas tarefas (fig. 6.2).

Ao clicar no botão para adicionar uma avaliação é exibida uma tela (fig. 6.8) com o formulário onde insere-se as informações sobre a avaliação. Nesta tela deve-se saber sobre dois parâmetros de fundamental importância na forma como a prova será aplicada: estratégia de aplicação e tipo de prova.

A estratégia de aplicação é a forma com que serão selecionadas as questões para aplicação do teste. As duas opções disponíveis no protótipo são ordem fixa e balanço de dificuldade. O funcionamento destas estratégias é explicado com maior detalhe na seção 5.4.

O tipo de prova significa como a prova será construída, se será montada pela seleção direta de cada questão na prova ou pela seleção de categorias, de onde as questões serão selecionadas conforme a estratégia.

Para adicionar uma questão ou categoria na prova, basta clicar no sinal “+” ao lado do item desejado. Para realizar a mudança de ordem das questões ou categorias podemos usar as setas para cima e para baixo localizadas ao lado do item. Para fazer a remoção de uma questão ou categoria da prova basta clicar no “x” ao lado do item.

Figura 6.8: Cadastro de avaliação por categorias

Editar avaliação ↻ ✕

Nome: Teste de informática básica 📅 25/10/2016 ▼

Estratégia de aplicação: Por ordem fixa ▼

Turma para aplicação: Turma de Teste ▼

Tipo de prova:

Por categorias

Por questões

Categorias		Ordem das categorias		
Informática básica	+	Informática básica	✕	^ ▼
— Memórias	+			
CPD	+	Informática básica	✕	^ ▼
— Conceitos Introdutórios	+			
— Ordenação por Inserção	+			
— Ordenação por Seleção	+			
— Ordenação por Trocas	+			

CANCELAR GRAVAR

Fonte: Elaborada pelo autor

6.2.8 Relatório de avaliações

Esta funcionalidade do sistema exibe relatórios sobre avaliações criadas pelo professor. Ao clicar em uma avaliação é exibida uma tela onde de listagem dos alunos que realizaram o teste com suas respectivas notas. Ao clicar em um aluno da lista pode-se ver um relatório das questões respondidas por ele, mostrando onde ele acertou ou errou. A figura 6.9 exibe um exemplo de relatório de avaliação de um aluno.

Figura 6.9: Relatório de avaliações

Avaliação		
Sobre arquivos, marque a alternativa correta:		
Respostas	Correta	Marcada pelo usuário
Um arquivo é uma coleção de registros	✓	✓
Registros lógicos podem conter diversos registros físicos, na maioria dos casos.		
A chave primária de um registro é um atributo que pode ser repetido no arquivo.		
A cada operação de E/S, somente um registro lógico é lido, na grande maioria dos casos.		
Registros lógicos sempre possuem tamanho variável		
São algoritmos de classificação por trocas:		
Respostas	Correta	Marcada pelo usuário
Heapsort e Quicksort		
Bubblesort e Insertion Sort		✓
...		

Fonte: Elaborada pelo autor

6.3 Atividades do aluno

As atividades do aluno são mais simples que as do professor, tendo apenas a opção de realizar avaliações. Uma avaliação pode ser iniciada ao clicarmos nela.

As questões são exibidas isoladamente para o aluno, cabendo a este selecionar a alternativa correta e clicar no botão “enviar” para ter sua correção e feedback retornados pelo sistema. Depois de enviar e obter o feedback o aluno pode prosseguir para a próxima questão clicando no botão “próxima”.

Ao finalizar a prova o aluno pode conferir suas questões clicando novamente na avaliação respondida. Será exibido um relatório como mostrado na figura 6.9.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou o sistema AvalWeb[®], um sistema que propõe uma solução de avaliações eletrônicas com o objetivo de gerar informações pedagógicas sobre o desempenho de turmas e alunos de forma a apoiar o educador no processo de ensino.

Foram apresentadas em ordem cronológica as implementações anteriores do sistema, mostrando suas diferenças de funcionalidades. Dentro de um contexto onde o sistema não recebeu novas atualizações percebeu-se a necessidade de uma reavaliação da arquitetura existente para a realizar a reconstrução do software, visando uma maior compatibilidade com as tecnologias e práticas existentes atualmente na indústria.

As mudanças propostas visaram a melhoria de requisitos funcionais e também de requisitos não-funcionais do sistema. Os requisitos funcionais sofreram algumas adições e remoções no protótipo implementado por questões de escopo de projeto. Os requisitos não-funcionais tiveram adições relevantes para que o sistema possa ser expandido e utilizado no futuro.

7.1 Aspectos técnicos

Este trabalho teve como um dos objetivos modernizar a arquitetura do sistema, para isso foi necessária a pesquisa e aprendizado de diversas tecnologias envolvidas neste processo.

As tecnologias escolhidas para a construção do sistema apresentaram resultados satisfatórios conforme os requisitos elicitados na forma de *user stories*. As maiores dificuldades envolvidas na execução do projeto estiveram ligadas às questões de paradigmas de programação e paradigmas de banco de dados propostos nestas tecnologias.

A linguagem de programação *Javascript* apesar de ser de aprendizado fácil, quando é utilizada em ambientes de grande quantidade de entrada e saída de dados demanda conhecimentos mais complexos e com muitas soluções diferentes para os mesmos problemas. Saber escolher a melhor solução para o projeto é uma tarefa difícil, principalmente para o programador principiante.

O aprendizado sobre as tecnologias de Front-end do sistema, como o *AngularJS* e o *Angular Material* têm uma curva de aprendizado mais amena que as tecnologias utilizadas no ambiente de Back-end, neste caso *Node.js* e *MongoDB*. No caso do *Node.js* é necessário aprender como utilizar a programação assíncrona para realizar as funções do

sistema que dependem de uma ordem de execução específica. No caso do MongoDB a maior dificuldade está na linguagem de consulta e inserção de dados, porém esta teve as suas dificuldades amenizadas pela utilização do framework *Mongoose* que possui uma interface própria para comunicação com a base de dados.

7.2 Trabalhos futuros

Este trabalho deixa em aberto algumas funcionalidades já propostas nos trabalhos anteriores e que não foram implementadas nesta versão. As possibilidades de funcionalidades são muitas, como podemos citar aqui, os tipos diferentes de questões além das alternativas de múltipla escolha, o levantamento de questões mais erradas pelos alunos, o acompanhamento do desempenho do aluno ao longo do tempo, a possibilidade da criação de provas mistas com questões e categorias, o tempo controlado para realização das questões, entre outras ideias que poderão ser implementadas de forma incremental ao protótipo inicial desenvolvido aqui.

Uma possibilidade interessante é a criação de métodos de correção de questões, criados na forma de estratégias programáveis. Dentro do âmbito das estratégias programáveis seria interessante a elaboração de estratégias de forma mais fácil dentro do próprio sistema. Isso poderia ser feito na forma de uma linguagem visual, como fluxogramas ou tipos de elementos gráficos.

Outra possibilidade interessante a ser investigada é verificar padrões para a implementação de APIs REST de forma que os serviços oferecidos possam ser utilizados por outros sistemas como as redes sociais. A possibilidade da integração de redes sociais e sistemas de ensino torna o sistema mais atraente ao público, facilitando a adoção e utilização.

A criação de um tipo de avaliação a qual poderia ser realizada de forma intermitente, ou seja, o aluno poderá responder algumas questões e deixar o restante para responder em outro momento. Este tipo de avaliação exige também a criação de um controle visual do progresso do aluno em relação à conclusão do teste. Este controle visual seria importante até mesmo para os tipos de avaliação tradicional, fornecendo para o aluno uma previsão do tempo que este levará para finalizar a prova.

A internacionalização do projeto, visando a tradução da interface do sistema é uma característica importante para longo prazo ao projeto, novamente visando atingir um maior público.

REFERÊNCIAS

- ANGULAR-MATERIAL. **Angular Material**. 2016. <<https://material.angularjs.org/latest/>>. [Online: acessado 4 de Outubro de 2016].
- ANGULARJS. **Data Binding**. 2010. <<https://docs.angularjs.org/guide/databinding>>. [Online: Acessado 29 de Outubro de 2016].
- ANGULARJS. **Understanding Controllers**. 2010. <<https://docs.angularjs.org/guide/controller>>. [Online: Acessado 29 de Outubro de 2016].
- ANGULARJS. **What is Angular?** 2010. <<https://docs.angularjs.org/guide/introduction>>. [Online: Acessado 29 de Outubro de 2016].
- CARDOSO, R. F. **AvalWeb - Sistema interativo para gerência de questões e aplicação de avaliações na Web**. 2001.
- CLASSEN, D. **Comparison of 4 popular JavaScript MV* frameworks (part 2)**. 2015. <<http://www.developereconomics.com/comparison-4-popular-javascript-mv-frameworks-part-2/>>. [Online - Acessado 5 de Outubro de 2016].
- COHN, M. **User stories applied**. first. [S.l.]: Addison Wesley, 2004.
- FIELDING, R. **Representational State Transfer (REST)**. 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. [Online: Acessado 31 de Outubro de 2016].
- FLANAGAN, D. **JavaScript: The Definitive Guide**. [S.l.]: O'Reilly, 2002.
- HAHN, E. M. **Express in Action**. [S.l.]: Manning publications, 2016.
- JSON.ORG. **Introducing JSON**. 2016. <<http://www.json.org/>>. [Online: Acessado 31 de Outubro de 2016].
- LUPCHINSKI, R. de L. F. **Desenvolvimento de uma Aplicação de Página-Única e Banco de Dados Não-Relacional para Organização e Controle de Eventos Esportivos**. 2015.
- MARCOTTE, E. **Responsive Web Design**. 2010. <<http://alistapart.com/article/responsive-web-design/>>. [Online: Acessado 9 de Novembro de 2016].
- MATERIAL-DESIGN. **Material Design**. 2016. <<https://material.google.com/>>. [Online: acessado 4 de Outubro de 2016].
- MONGODB. **Introduction to MongoDB**. 2016. <<https://docs.mongodb.com/manual/introduction/>>. [Online: Acessado 31 de Outubro de 2016].
- MONGOOSE. **Getting Started**. 2016. <<http://mongoosejs.com/docs/index.html>>. [Online: Acessado 31 de Outubro de 2016].
- MONGOOSE. **Mongoose Schemas**. 2016. <<http://mongoosejs.com/docs/guide.html>>. [Online: Acessado 5 de Novembro de 2016].

- MONGOOSE. **Mongoose Schemas**. 2016. <<http://mongoosejs.com/docs/schemas.html>>. [Online: Acessado 1 de Novembro de 2016].
- MORAIS, C.; LIMA, J. V. de; FRANCO, S. K. **AvalWeb - Sistema interativo para gerência de questões e aplicação de avaliações na Web**. 2005.
- MOZILLA. **JavaScript**. 2015. <<https://developer.mozilla.org/en-US/docs/Web/JavaScript/>>. [Online: Acessado 20 de Setembro de 2016].
- MOZILLA. **Ajax Primeiros Passos**. 2016. <https://developer.mozilla.org/pt-BR/docs/AJAX/Getting_Started/>. [Online: Acessado 20 de Setembro de 2016].
- NODE.JS. **About Node.js®**. 2016. <<https://nodejs.org/en/about/>>. [Online: Acessado 5 de Outubro de 2016].
- NPM. **Packages and Modules**. 2016. <<https://docs.npmjs.com/how-npm-works/packages/>>. [Online: Acessado 5 de Novembro de 2016].
- NPM. **What is npm?** 2016. <<https://docs.npmjs.com/getting-started/what-is-npm/>>. [Online: Acessado 5 de Novembro de 2016].
- OVERFLOW, S. **Stack Overflow Developer Survey Results 2016**. 2016. <<http://stackoverflow.com/research/developer-survey-2016/>>. [Online: acessado 30 de Outubro de 2016].
- PASSPORT. **Passport documentation**. 2016. <<http://passportjs.org/docs/>>. [Online: Acessado 31 de Outubro de 2016].
- PRIMO, G. **User Stories – O que são? Como Usar?** 2011. <<http://blog.myscrumhalf.com/2011/10/user-stories-o-que-sao-como-usar/>>. [Online: Acessado 31 de Outubro de 2016].
- SCHLUETER, I. **AngularJS**. 2014. <<http://www.dwmkerr.com/promises-in-angularjs-the-definitive-guide/>>. [Online: Acessado 10 de Novembro de 2016].
- STRONG-LOOP. **What Makes Node.js Faster Than Java?** 2014. <<https://strongloop.com/strongblog/node-js-is-faster-than-java/>>. [Online: Acessado 5 de Outubro de 2016].
- TABLELESS. **O que é client-side e server-side?** 2012. <<http://tableless.github.io/iniciantes/manual/obasico/o-que-front-back.html>>. [Online: Acessado 30 de Outubro de 2016].
- TORRES, D. de O. **Sistema de gerência de questões e respostas : AvalWeb 2.0**. 2009.
- VERA, H. Data modeling for nosql document-oriented databases. **Proceedings SIMBig**, 2015.
- W3C. **HTML & CSS**. 2016. <<https://www.w3.org/standards/webdesign/htmlcss/>>. [Online: acessado 30 de Outubro de 2016].
- WIKIPEDIA. **Single Page Application**. 2016. <https://en.wikipedia.org/wiki/Single-page_application/>. [Online: Acessado 31 de Outubro de 2016].