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

GABRIEL MASCHKE FERREIRA

SIR-EDU: Sistema Integrado de Recursos Educacionais para a Gestão do Acompanhamento de Alunos com Necessidades Especiais

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

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2017

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Rui Vicente Oppermann
Vice-Reitor: Profa. Jane Fraga Tutikian
Pró-Reitor de Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Profa. Carla Maria dal Sasso Freitas
Coordenador do Curso de Ciência de Computação: Prof. Prof. Raul Fernando Weber
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

SUMÁRIO

LISTA DE FIGURAS **5**
LISTA DE TABELAS **6**
LISTA DE ABREVIATURAS E SIGLAS **7**
RESUMO **8**
ABSTRACT **9**
1 INTRODUÇÃO **10**
2 TRABALHOS RELACIONADOS **11**
2.1 Moodle **11**
3 TECNOLOGIAS UTILIZADAS E ARQUITETURA DO SISTEMA **14**
3.1 Arquitetura do Sistema **14**
3.2 Front-End **15**
3.2.1 HTML 15
3.2.2 CSS 15
3.2.3 Javascript e ECMAScript 16
3.2.4 AJAX e JSON 16
3.2.5 Single-Page Aplication 17
3.2.6 ReactJS, Flux e Redux 17
3.3 Aplicativo Híbrido Mobile **19**
3.3.1 Apache Cordova 19
3.3.2 Ionic 20
3.4 Back-End **20**
3.4.1 Node.js e npm 21
3.4.2 REST 22
3.4.3 Express 23
3.4.4 Autenticação com JSONWebToken 23
3.4.5 MongoDB 24
3.4.6 GridFS 25
3.4.7 Mongoose 25
4 PROJETO DO SISTEMA **27**
4.1 Funcionalidades Necessárias **27**
4.1.1 User Stories do sistema Web 27
4.1.2 User stories do aplicativo móvel 28
4.2 Modelagem do Banco de Dados **29**
4.2.1 Entidade User 29
4.2.2 Entidade Student 29
4.2.3 Entidade Judgement 31
4.2.4 Entidade File 32
4.2.5 Entidade School 33
4.2.6 Diagrama de Entidades 34
4.3 API REST **35**
4.3.1 User 36
4.3.2 Students 37
4.3.3 Files 38
4.3.4 Judgement 39
4.3.5 School 40
5 IMPLEMENTAÇÃO DO PROJETO **41**
5.1 Back End do Sistema **41**
5.1.1 Configuração com npm 42

5.1.2 Estrutura da API do sistema 42
5.1.3 Autenticação 43
5.1.4 Definição das entidades com *schemas* do Mongoose 44
5.1.5 Definição das rotas da API com o Express 45
5.2 Sistema web 46
5.3 Aplicativo Mobile 48
6 INTERFACES DE NAVEGAÇÃO 50
6.1 Sistema web 50
6.1.1 Tela de login 50
6.1.2 Tela de cadastro de usuário 50
6.1.3 Cadastro de aluno 51
6.1.4 Tela da lista de alunos 55
6.1.5 Menu do aluno 55
6.1.6 Tela de envio de arquivos 56
6.1.7 Tela de registro de pareceres 57
6.1.8 Linha do tempo 59
6.2 Aplicativo mobile 61
6.2.1 Tela de login 61
6.2.2 Lista de alunos 62
6.2.3 Menu lateral 62
6.2.4 Cadastro de alunos 63
6.2.5 Lista de arquivos 64
6.2.6 Tela de criação e adição de mídias 65
6.3 Avaliação dos Sistemas 67
7 CONCLUSÃO 68
REFERÊNCIAS 70

LISTA DE FIGURAS

Figura 2.1 Tela de visualização no papel de um professor no Moodle 12

Figura 3.1 Diagrama da arquitetura Flux 14
Figura 3.2 Diagrama da arquitetura Flux 18
Figura 3.3 Exemplo de aplicação Ionic 21

Figura 4.1 Diagrama das entidades da base de dados NoSQL 34

Figura 5.1 Estrutura de arquivos no servidor 41
Figura 5.2 Estrutura de arquivos da API 43
Figura 5.3 Trecho de código de autenticação 44
Figura 5.4 Schema da entidade student 45
Figura 5.5 Exemplo de rota do Express 46
Figura 5.6 Estrutura de arquivos do sistema web 46
Figura 5.7 Exemplo do componente *DialogModal* em *React* 47
Figura 5.8 Declaração de uma rota 47
Figura 5.9 Estrutura de arquivos do sistema web 48
Figura 5.10 Código HTML da tela de login da aplicação utilizando Ionic 49

Figura 6.1 Tela de login 51
Figura 6.2 Tela de cadastro de usuário 51
Figura 6.3 Etapa 1: Dados do aluno 52
Figura 6.4 Etapa 2: Atividades extraclasse 53
Figura 6.5 Etapa 3: Necessidades educacionais especiais 53
Figura 6.6 Etapa 4: Finalização cadastro 54
Figura 6.7 Lista de alunos 55
Figura 6.8 Tela do menu do aluno 56
Figura 6.9 Tela de envio de arquivos 57
Figura 6.10 Exemplo de cadastro de uma foto 57
Figura 6.11 Exemplo de criação de parecer 58
Figura 6.12 Linha do tempo de um aluno 59
Figura 6.13 Visualização de um parecer 60
Figura 6.14 Visualização de um arquivo 60
Figura 6.15 Tela de login 61
Figura 6.16 Tela da lista de alunos 62
Figura 6.17 Menu de opções 63
Figura 6.18 Tela da lista de arquivos de um aluno 64
Figura 6.19 Tela de registro de arquivos 66

LISTA DE TABELAS

Tabela 4.1 Tabela de operações para o recurso user 36
Tabela 4.2 Tabela de operações para o recurso students 37
Tabela 4.3 Tabela de operações para o recurso files 38
Tabela 4.4 Tabela de operações para o recurso judgement 39
Tabela 4.5 Tabela de operações para o recurso school 40

LISTA DE ABREVIATURAS E SIGLAS

AEE	Atendimento Educacional Especializado
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
BSON	Binary JSON
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
MVC	Model View Controller
NEE	Necessidades Educacionais Especiais
NoSQL	Non Structured Query Language
REST	Representational State Transfer
SPA	Single Page Application
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

RESUMO

O objetivo deste trabalho é de desenvolver um protótipo de sistema que se utiliza do ambiente Web com a arquitetura de *Single-page Application*, em conjunto com um aplicativo para celular, que juntos possibilitam professores especializados a acompanhar e registrar atividades pedagógicas realizadas em escolas, universidades, ou qualquer ambiente educacional, visando a análise de alunos portadores de necessidades especiais. O registro dessas atividades é feito em formato textual (formulário) e pode ser acompanhado de imagens ou vídeos. O sistema ainda consiste de uma *API REST* responsável por integrar a versão Web e móvel para realizar o cadastro de alunos, professores/pesquisadores, instituições de ensino, fotos e vídeos. A partir dessa documentação e dados armazenados, poderão ser gerados relatórios como, por exemplo: planos de atendimento individual, adequações curriculares, programação curricular, pareceres do professor, entre outros.

Palavras-chave: API REST. Single-Page Application. Aplicativo de celular. Web.

SIR-EDU: Integrated System of Educational Resources for accompaniment of students with Special Needs

ABSTRACT

The objective of this work is to develop a system that uses the web environment based on the Single-Page Application architecture, in conjunction with a mobile application, to allow specialized teachers to monitor and record pedagogical activities carried out in schools, universities, or any educational environment, aiming at the analysis of students with special needs. The registration of these activities is done in textual format (forms) and can be accompanied by images or videos. Besides that, the system consists of a REST API responsible for integrating the Web and mobile versions. The API provides the functionality for registering students, teachers/researchers, educational institutions, photos, and videos. From this documents and stored data, reports can be generated, like for example individual attendance plans, curricular adaptations, curricular programming, teacher opinions, among others.

Keywords: REST API, Single-Page Application, Web, Mobile Application.

1 INTRODUÇÃO

Nas escolas o Atendimento Educacional Especializado (AEE) dá-se nas Salas de Recursos, onde profissionais especializados acompanham os alunos com Necessidades Educacionais Especiais (NEE) e apoiam os professores de sala de aula regular, no processo de aprendizagem destes alunos. É necessário que sejam gerenciados uma série de documentos e protocolos que são gerados a partir de atendimentos, intervenções e observações com o aluno.

Ferramentas de captura de vídeos e fotos são amplamente utilizadas pelos professores especializados para a análise do processo de aprendizagem dos alunos com NEE. Porém, atualmente esses registros são armazenados na memória da câmera/celular do professor, e, por fim, são transferidas para uma pasta de cada aluno no sistema de arquivos do computador. O mesmo processo acontece com os documentos gerados em editores de texto. Todos esses dados são acessíveis através de um computador localizado dentro da Sala de Recursos e é difícil manter o controle e segurança de acesso aos mesmos.

Tendo em vista essa diversidade de informações e seu complexo gerenciamento, foi identificada a necessidade de se desenvolver um sistema capaz de armazenar e gerenciar essas informações para cada aluno, e consequentemente, torná-las acessíveis de maneira rápida e prática através de qualquer dispositivo.

Nesse contexto, o objetivo principal deste trabalho é de dar início ao desenvolvimento do Sistema Integrado de Recursos Educacionais para o Acompanhamento de Alunos com Necessidades Especiais (SIR-EDU), criando os protótipos de sistema Web e aplicativo de celular, além de um *Back-End* com uma API de acesso aos dados.

O trabalho está organizado em 6 capítulos. No próximo capítulo serão identificados os requisitos fundamentais do sistema a ser desenvolvido, e também serão explicadas as ferramentas utilizadas e as decisões de projeto para seu uso, juntamente com a arquitetura utilizada. Em seguida, no terceiro capítulo será descrita a modelagem do sistema, apresentando as entidades do banco de dados e a API de acesso aos recursos. O quarto capítulo exibe os detalhes técnicos sobre a implementação do projeto. No quinto capítulo serão apresentados os resultados através das telas dos protótipos desenvolvidos. Finalmente, no sexto e último capítulo, serão revisitados os objetivos iniciais deste trabalho, discutindo sobre sua contribuição e sugestões para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Este capítulo apresenta uma análise dos trabalhos relacionados já existentes. A maioria das soluções que podem ser encontradas hoje em dia possuem funcionalidades semelhantes, porém estão mais associadas ao ensino a distância ou a objetos de aprendizagem. Por causa disso, essas aplicações possuem um propósito diferente deste trabalho. Como exemplo dessas aplicações, foi escolhido o sistema Moodle para ser avaliado e comparado com as necessidades de um professor especializado.

2.1 Moodle

O Moodle é uma plataforma virtual de aprendizagem que constitui-se em um sistema de administração de atividades educacionais destinado à criação de comunidades *on-line*, em ambientes virtuais voltados para a aprendizagem colaborativa. Permite, de maneira simplificada, a um estudante ou a um professor integrar-se, estudando ou lecionando, num curso *on-line* à sua escolha.

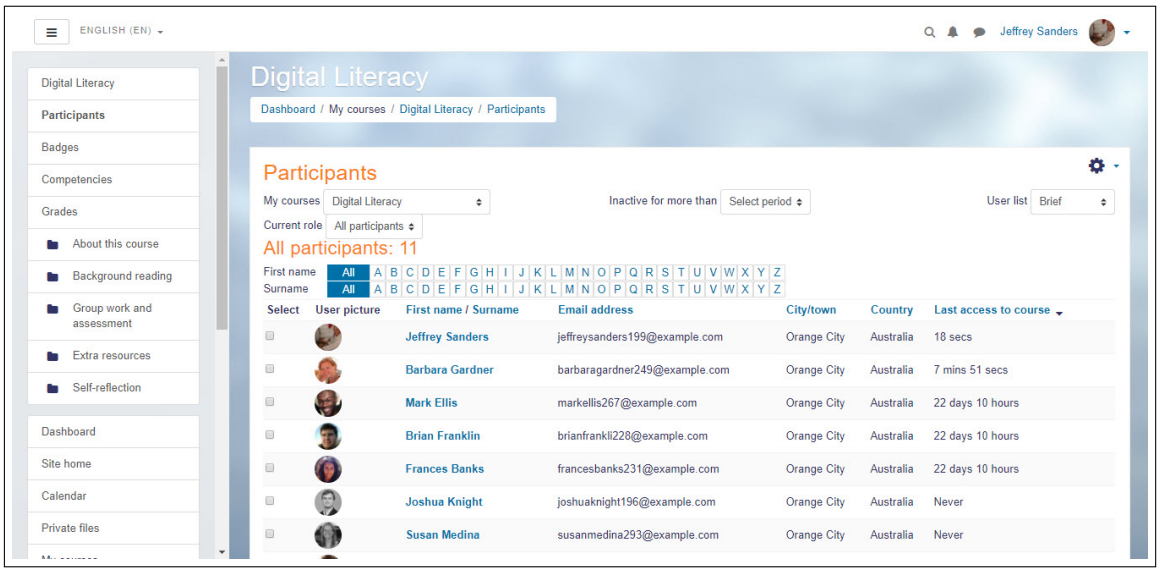
Abaixo são listadas as principais funcionalidades do Moodle:

- Produção e distribuição de conteúdo;
- Compartilhamento de conteúdo entre instituições;
- Gestão do ambiente virtual de aprendizagem;
- Gestão de alunos por parte do professor;
- Realização de avaliações de alunos;
- Suporte tecnológico para a disponibilização de conteúdos de acordo com um modelo pedagógico da instituição;
- Disponibilização dos cronogramas de disciplinas;
- Controle de acessos;
- Atribuição de notas.

Com base nisso é possível observar que o Moodle pode ser amplamente utilizado no suporte à educação dentro da sala de aula regular ou através de classes de ensino à

distância, dando ferramentas para o professor e o aluno interagirem virtualmente através da plataforma. Porém, seu uso está focado na divisão de disciplinas e controle dos conteúdos a serem passados para todos os alunos de uma classe, e se um desses alunos for um aluno com necessidades educacionais especiais (NEE), um cronograma específico com objetivos individuais deverá ser traçado previamente. Esse cronograma deverá ser adaptado para as necessidades do aluno e deverá ser criado em conjunto com um professor especializado, pois é ele que detém o conhecimento para avaliar as necessidades do aluno e adaptações necessárias de acordo com o acompanhamento dentro da sala de recursos. Esse processo de avaliação e acompanhamento do aluno com NEE é um processo que deve idealmente ocorrer antes do começo de uma disciplina, e deve ser incrementado continuamente para que o professor especializado consiga dar o suporte para os professores de sala de aula regular com relação aos alunos NEE.

Figura 2.1: Tela de visualização no papel de um professor no Moodle



Fonte: Elaborada pelo autor

Apesar dessas ferramentas de objetos de aprendizagem possuírem mecanismos para gestão individual de alunos e geração de relatórios, o seu foco é voltado para a ministração do curso em si e não no atendimento individual de alunos com NEE, além disso, adaptar a plataforma para se adequar ao processo do Atendimento Educacional Especializado (AEE) de uma instituição pode ser custoso e não conter as funcionalidades necessárias. A ideia é que o projeto deste trabalho seja utilizado em conjunto com ferramentas como o Moodle para que as disciplinas e a instituição de ensino se adaptem ao aluno especial. Visto que todo esse processo de avaliação do aluno é feito por um pro-

fessor especializado, os próximos capítulos descrevem o desenvolvimento de um sistema que disponibilizará ferramentas de apoio a esses professores com relação a gestão e intervenções pedagógicas de alunos com NEE e também na geração de relatórios para que as instituições consigam se adaptar ao aluno em questão.

3 TECNOLOGIAS UTILIZADAS E ARQUITETURA DO SISTEMA

Este capítulo apresenta as ferramentas utilizadas para o desenvolvimento do projeto. O capítulo está dividido em quatro seções: *Front-End*, Aplicativo Híbrido *Mobile*, *Back-End* e Arquitetura do Sistema.

A primeira seção trata da explicação da arquitetura do sistema utilizada. A segunda seção aborda as tecnologias da camadas de cliente da aplicação, e, neste caso, são explicadas as características e tecnologias do *Front-End* do sistema.

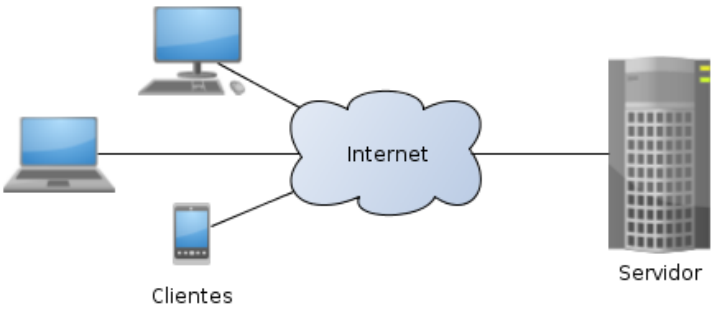
A terceira seção é sobre as tecnologias e estrutura do aplicativo híbrido móvel.

Por fim são apresentadas as tecnologias do *Back-End* no servidor da aplicação e persistência de dados.

3.1 Arquitetura do Sistema

Este projeto como um todo é baseado na arquitetura Cliente-Servidor. Esse modelo computacional possui uma estrutura de aplicação distribuída onde são definidas duas camadas, a de servidor, que fornece e gerencia os recursos do sistema, e a camada de cliente, que consome e faz a requisição desses recursos. Toda a comunicação entre as camadas ocorre através de uma rede, nesse caso, a Internet.

Figura 3.1: Diagrama da arquitetura Flux



Fonte: https://pt.wikipedia.org/wiki/Cliente-servidor

Neste trabalho o sistema web, que executa no navegador do usuário, e o aplicativo de celular são instâncias da camada Cliente que se comunicam via HTTP com o Servidor. O servidor é responsável por atender as requisições dos clientes, e é nele que são implementadas a API REST, construída com Node.js e Express, e a persistência de dados com

o MongoDB.

3.2 Front-End

Esta seção tem como objetivo explicar o conceito e tecnologias associadas ao desenvolvimento *Front-End* do Sistema Web. Nas suas subseções são descritas as tecnologias que o compõem.

O *Front-End* de um sistema está atrelado às tecnologias executadas no lado do cliente, ou seja, no navegador de internet. Esta camada é responsável por mostrar a interface de interação com o usuário do sistema, gerenciar a entrada de dados e atualização de informações recebidas pelo servidor da aplicação. As tecnologias como o HTML, JavaScript e CSS são utilizadas pelos desenvolvedores para criar os elementos visuais do *Front-End* de uma aplicação

3.2.1 HTML

HTML é a abreviação de *HyperText Markup Language*. HTML é uma linguagem de marcação utilizada para criação e estruturação dos elementos que compõem o conteúdo de uma página Web. Ela foi definida como o padrão na criação de aplicações web e é interpretada por todos os navegadores modernos. Atualmente, HTML está na quinta versão e conta com o suporte nativo para os últimos formatos de multimídia como vídeos, áudios e imagens (MOZZILA, 2017b). Em conjunto com ferramentas de estilo de elementos como o CSS e com o JavaScript, que gerencia o comportamento da página, o HTML é uma poderosa ferramenta para construção de sistemas Web.

3.2.2 CSS

Cascading Style Sheets ou CSS é uma linguagem de estilos utilizada para descrever como os elementos HTML devem ser apresentados e formatados numa página Web. Para o desenvolvimento deste trabalho, foi utilizado o *framework open-source* de *Front-End* Bootstrap(<http://getbootstrap.com/>), que consta com uma variedade de componentes e estilos criados por designers de web, que facilitam a criação de páginas HTML.

3.2.3 Javascript e ECMAScript

A linguagem de programação JavaScript (JS) é uma linguagem multiparadigma que suporta orientação a objetos, programação funcional e programação imperativa. Inicialmente a JS foi idealizada para ser executada na camada de cliente da aplicação, interagindo com os elementos de uma página Web. Porém, com o avanço das tecnologias, suporte da comunidade e evolução da linguagem, através do NodeJS também é possível desenvolver aplicações que são executadas na camada de servidor de um sistema (W3C, 2012).

JavaScript segue a especificação *ECMAScript* para padronização de implementação pelos navegadores modernos. Atualmente os navegadores suportam completamente o padrão ECMAScript 5.1 (MOZZILA, 2017c), neste trabalho foi utilizada a sexta versão, que consta com uma série de novas funcionalidades úteis que ainda não são suportadas por todos navegadores. Por essa razão, para a construção do ambiente de desenvolvimento, foi utilizada a ferramenta *Webpack* para empacotamento e agrupamento de arquivos JS (WEBPACK, 2017), que é capaz de pré-processar um arquivo e transformar os comandos que usam a sintaxe ECMAScript6 para ECMAScript5.

Neste trabalho, JavaScript foi utilizada tanto na parte de *Front-End* como na parte de *Back-End*, ou seja, a mesma linguagem utilizada pelo sistema web, o aplicativo híbrido móvel e o servidor. Do ponto de vista de desenvolvimento, é uma grande vantagem para o programador poder codificar diferentes camadas da aplicação na mesma linguagem, pois não há a curva de aprendizagem que o conjunto de outras linguagens exigiria.

3.2.4 AJAX e JSON

Asynchronous JavaScript And XML ou AJAX é uma ferramenta utilizada para explorar a característica assíncrona do JavaScript para comunicação e troca de dados com servidores sem a necessidade de recarregar a página (MOZZILA, 2017a).

Neste trabalho, toda a troca de informações entre o cliente e servidor é feita no formato *JavaScript Object Notation* (JSON). Um arquivo nesse formato é composto por um par de chave e valor, fácil para humanos entenderem e para máquinas realizarem a leitura do documento. Em muitas linguagens de programação o arquivo JSON é interpretado como um objeto (JSON.ORG, 2017), o que torna fácil a sua manipulação dentro do código.

3.2.5 Single-Page Aplication

Single-page application ou SPA, é um termo normalmente utilizado para descrever aplicações que são criadas especificamente para a Web. Essas aplicações são acessadas pelo navegador mas têm o diferencial de se assemelharem ao comportamento dinâmico de uma aplicação nativa móvel ou *desktop*, ou seja, sem uma grande quantidade de recarregamentos da página para realizar a navegação (CODESCHOOL, 2017).

Essa característica faz com que o website SPA utilize muitas chamadas AJAX para atualizar seu conteúdo dinamicamente, como resultado disso, o processo dinâmico de renderização e de roteamento das páginas fica a cargo do cliente.

No fluxo usual de uma SPA, o navegador faz o primeira chamada para o servidor requisitando a página a ser acessada, o servidor responde com todo o código JavaScript e apenas um `index.html`. O *client-side* fica responsável por *renderizar* e controlar as atualizações da página. Essa abordagem difere da *server-side rendering*, onde o servidor retorna as páginas com a informação necessária para o cliente, gerando os recarregamentos mencionados anteriormente.

Muitos *frameworks* JavaScript se basearam nesse conceito para ajudar na criação de SPAs, dentre eles são mais famosos o AngularJS e o ReactJs. Para o desenvolvimento deste trabalho, o *framework* escolhido foi o ReactJS, desenvolvido pelo Facebook.

3.2.6 ReactJS, Flux e Redux

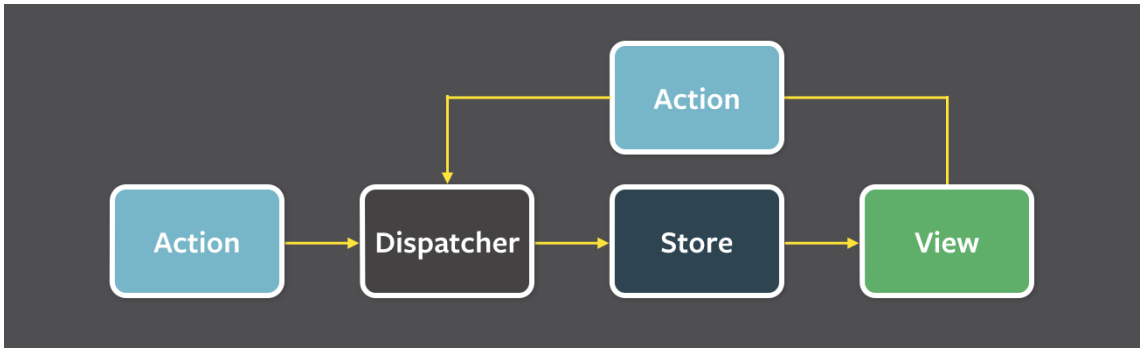
ReactJS é um *framework* desenvolvido pelo Facebook para a criação de interfaces de usuário interativas. Sua arquitetura é baseada em componentes que podem ser reusáveis em todo o código, nessa arquitetura cada componente possui *Single Responsability*, ou seja, cada um é responsável por gerenciar seu estado e atualizar sua camada de visualização. Para facilitar a atualização dos componentes, React implementa uma *Virtual DOM* que atualiza o DOM verdadeiro apenas quando necessário. Nesse caso, o componente só será atualizado caso uma chamada AJAX tenha retornado, ou algum outro componente tenha disparado sua atualização. A *Virtual DOM* analisará as suas diferenças com a DOM real, e então é atualizado somente o necessário sem *renderizar* a página inteira novamente (REACT, 2017).

Como o ReactJS é um *framework* para desenvolver SPAs, é necessário um sistema de rotas de URLs internas para a aplicação, o *React Router* é o responsável gerenciar o

estado do sistema de rotas. A cada rota é associado um componente controlador de uma tela, dentro deste componente pai é possível ter outros componentes. Esses componentes mais externos são chamados de *Container Components*.

O conceito de *Component Based Architeture* difere do tradicional MVC onde cada *controller* é responsável por gerenciar o conteúdo de uma *view*. Tendo em vista essa complexidade de atualização de estados desencadeada por cada componente na aplicação, o Facebook idealizou um padrão de arquitetura chamado *Flux* para complementar o desenvolvimento das aplicações, utilizando o que eles chamam de *Unidirectional Data Flow*. No Flux tem-se uma estrutura de $Views$ (Componentes), $Actions$, um $Dispatcher$ e $Stores$ (FLUX, 2015).

Figura 3.2: Diagrama da arquitetura Flux



Fonte: https://facebook.github.io/flux/docs/in-depth-overview.html

Na Figura 3.1 é exemplificado o funcionamento da arquitetura Flux. Todos os dados passam pelo *dispatcher* (despachante), que é uma espécie de *hub* central que processa as *actions* (ações). As ações são criadas pelas interações dos usuários com as *views* (interfaces) e carregam o tipo de dado que será alterado, por exemplo: "o usuário clicou no botão de login". O despachante então envia essas ações para todas as *stores* (armazenadores de estados). Os armazenadores de estados contêm o estado e a lógica da aplicação, seu papel é similar ao *Model* (Modelo) do MVC, porém um armazenador de estado pode gerenciar o estado de diversos componentes associados a ele. Se a ação é relevante para o armazenador de estado, ela emite um evento de mudança para alertar os componentes que contém as interfaces informando que alguma mudança nos dados ocorreu. Os componentes que detectarem essa mudança são responsáveis por atualizar seu estado e propagar o sinal de mudança nos dados do armazenador de estados para todos os seus filhos (FLUX, 2015).

O ciclo do processamento de ações ocorre enquanto houver ações para serem pro-

cessadas durante o tempo de vida da aplicação. Caso o usuário pare de interagir com a interface, é provável que o ciclo fique ocioso esperando por alguma mudança.

O Redux é uma biblioteca que implementa a arquitetura Flux e permite a integração com o React, ele é o responsável por gerenciar o estado da aplicação (*stores*), porém há algumas modificações perante o padrão proposto pelo Facebook. Em vez de utilizar múltiplos armazenadores de estados, é utilizado o conceito de *Reducer* (Redutor), o redutor transforma todas armazenadores de estados da aplicação em apenas um, criando um único estado da aplicação. Assim os componentes se conectam as partes do estado que lhes interessam, sem a necessidade de escutar aos eventos de mudanças de diferentes armazenadores (REDUX, 2017)).

3.3 Aplicativo Híbrido Mobile

Esta seção explica o conceito de Aplicativo Híbrido *Mobile* (móvel). As suas subseções apresentam as tecnologias utilizadas na construção do aplicativo.

As Aplicações híbridas são construídas de maneira similar a um sistema web. Elas utilizam a mesma combinação de tecnologias como o HTML, JavaScript e CSS, ou seja, as mesmas tecnologias que são utilizadas no *Front-End* de uma aplicação também podem ser utilizadas para a criação de um aplicativo móvel. A grande diferença é que com o auxílio de ferramentas como o *Apache Cordova* é possível executar o aplicativo híbrido em múltiplas plataformas como se ele fosse uma *app* nativa.

3.3.1 Apache Cordova

O Apache Cordova é um *framework open-source* para desenvolvimento *mobile*. Ele é o responsável por habilitar o programador a utilizar tecnologias características da web como HTML, CSS e JavaScript para o desenvolvimento de aplicativos multiplataforma. As aplicações executam num ambiente chamado WebView, que *renderiza* as páginas construídas com as tecnologias web e permite a comunicação com as APIs desenvolvidas pelo Apache Cordova. Essas APIs são disponibilizadas na linguagem JavaScript para os programadores, provendo serviços que dão acesso às funcionalidades nativas do aparelho como sensores, status da rede, câmera, etc (CORDOVA, 2017). Além disso, o *framework* também disponibiliza a função de empacotamento dos arquivos de código para

gerar um aplicativo nativamente instalável para os dispositivos Android, IOS ou Windows Phone, criando um ícone de atalho no menu do sistema operacional.

3.3.2 Ionic

O Ionic é outro *framework* para desenvolvimento de aplicações híbridas, porém seu foco é voltado especialmente para a construção da interface do aplicativo. Ele disponibiliza uma série de componentes como menus, listas e botões construídos em HTML, CSS e JavaScript que se assemelham em funcionalidade e aparência aos componentes nativos de cada plataforma (IONIC, 2017). O Apache Cordova é utilizado internamente pelo Ionic para a instalação nativa do aplicativo e para utilização dos serviços de APIs.

Neste projeto, a utilização do Ionic possibilitou uma rápida construção da interface do sistema, pois seus componentes são facilmente adicionados na composição de uma página HTML, sem a necessidade de muitos ajustes por parte do programador. A funcionalidade de cada componente é gerenciada via JavaScript e chamadas AJAX. A integração com os *plugins* do Apache Cordova abstrai o processo de comunicação com as camadas nativas do sistema operacional, e isso permite que programadores sem experiência com aplicações nativas realizem operações complexas que normalmente exigiriam conhecimentos avançados da plataforma-alvo.

Na Figura 3.2 tem-se um exemplo de aplicação que utiliza o componente de lista do Ionic. Pode-se observar que a interface do sistema é muito parecida com uma aplicação nativa Android.

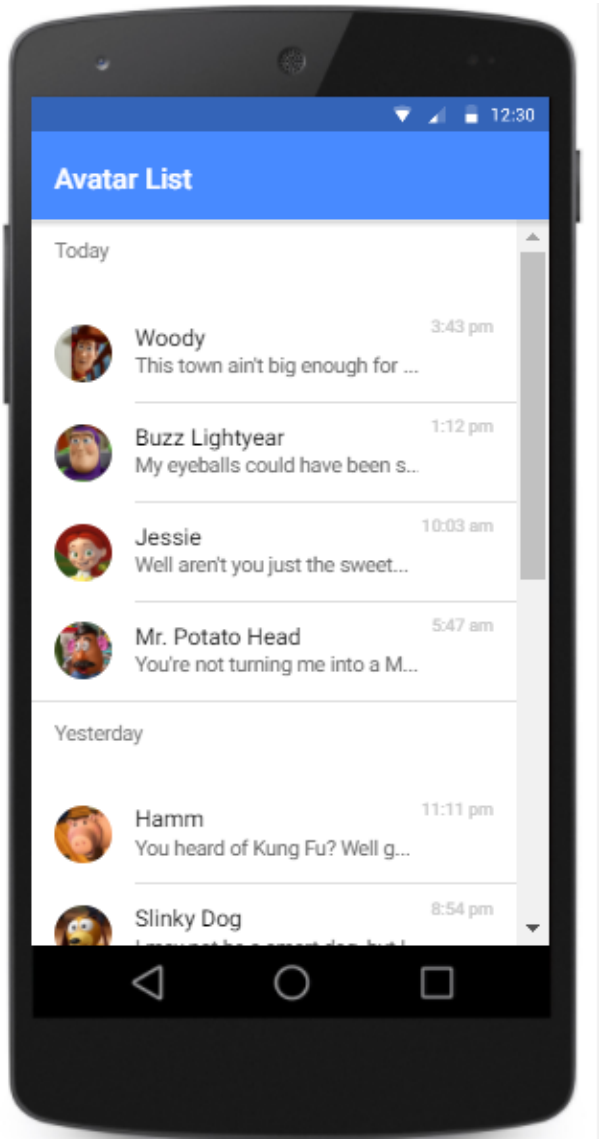
3.4 Back-End

Esta seção aborda as tecnologias associadas à camada de *Back-End* do projeto.

O *Back-End* de uma aplicação se refere às partes do sistema que o usuário não é capaz de ver, mas que são essenciais no funcionamento de um sistema. Num sistema web onde temos geralmente uma arquitetura cliente-servidor, o servidor é o responsável por realizar autenticação, processar dados informados pelo usuário provenientes do *Front-End*, escalar o sistema, servir conteúdo, atualizar informações, persistência de banco de dados, etc.

Na arquitetura deste projeto utiliza-se as seguintes tecnologias: *Node.js* e *Express*

Figura 3.3: Exemplo de aplicação Ionic



Fonte: https://ionicframework.com/docs/components

para fornecer serviços da *API REST* de comunicação com a camada cliente, e *MongoDB* para a persistência de dados.

3.4.1 Node.js e npm

Node.js é um ambiente de execução JavaScript construído em cima da *Engine V8* do Chrome. Essa *engine* de alta performance é responsável por compilar e executar códigos JavaScript, gerenciar alocação de memória para os objetos e também possui um sistema de *Garbage Collector* para objetos que não estão sendo mais usados (CHROME,

2017).

O Node.js foi criado para construção de aplicações de rede altamente escaláveis, ele utiliza um modelo *single threaded* orientado a eventos assíncronos e toda comunicação de entrada e saída é não bloqueante, ou seja, enquanto são performadas operações *I/O*, múltiplas conexões podem ser gerenciadas concorrentemente. Desta forma, como o processo nunca é bloqueado, sistemas escaláveis são fáceis de se alcançar com o Node.js (NODE.JS, 2017).

Neste projeto, o Node.js foi utilizado na camada de servidor da aplicação, sendo responsável por atender as requisições da camada cliente e por disponibilizar a API de acesso aos recursos e manipulação de dados.

O *npm* é um sistema gerenciador de pacotes para JavaScript, com ele é possível compartilhar e reutilizar códigos/bibliotecas, além de gerenciar as dependências de um projeto (NPM, 2017). Com o *npm* é fácil de compartilhar e manter o código entre equipes, com poucos comandos como o `npm install` é possível resolver todas as dependências necessárias para o sistema funcionar, sem muito esforço por parte do programador.

3.4.2 REST

Representational state transfer (REST) é um estilo de arquitetura para construção de sistemas distribuídos na internet.

A arquitetura REST é geralmente utilizada na construção de APIs para serviços web e é frequentemente associada ao protocolo HTTP para prover interoperabilidade dos sistemas. Há uma série de restrições para a API de um sistema ser considerada *RESTful*, entre elas, o sistema precisa ter uma arquitetura cliente-servidor, não manter estado e ter uma interface uniforme através de URIs. Pelo fato do sistema não possuir estados, cada requisição do cliente para o servidor precisa conter toda informação necessária para ser entendida, com isso a arquitetura garante as propriedades de visibilidade, escalabilidade e confiabilidade (FIELDING, 2000).

Neste projeto a API não cumpre totalmente a especificação REST, mas implementa seus principais princípios. Os recursos são expostos na forma de URIs, é utilizado o formato JSON para a representação e comunicação de dados, toda comunicação com o servidor ocorre através dos principais métodos do protocolo HTTP (GET, POST, PUT, DELETE), e o servidor não mantém estado entre as requisições.

3.4.3 Express

O *Express* é um *framework* para Node.js criado para facilitar a construção de APIs REST para sistemas Web. Ele consta com uma série de funcionalidades para ajudar o programador na criação de um servidor Node.js, também disponibiliza um robusto sistema de roteamento para processar requisições do *Front-End* da aplicação. Esse sistema de roteamento foi criado para processar e responder requisições correspondentes ao seu método do protocolo HTTP (EXPRESS, 2017).

Este *framework* ainda disponibiliza uma ferramenta chamada *middleware*. Sua arquitetura de uma aplicação é baseada em uma série de chamadas de funções de *middleware*. As funções de *middleware* têm o papel de executar comandos intermediários à uma requisição HTTP. Por exemplo, quando chega uma requisição de autenticação para API, após o método de entrada na rota, um *middleware* é executado para processar a autenticação, verificando se o usuário existe no banco de dados e respondendo a requisição para o cliente com a resolução apropriada.

Neste projeto o Express tem um papel chave, pois é com ele que são definidas todas as rotas da API REST, além de processar as requisições a recursos e servir o conteúdo estático da aplicação como páginas HTML, ícones e os *source codes* JavaScript.

3.4.4 Autenticação com JSONWebToken

JSON Web Token ou JWT é um padrão que define uma maneira compacta e segura de transmitir informações entre duas partes no formato de um objeto JSON. Esse objeto pode ser verificado e confiado, pois é assinada digitalmente. A assinatura pode ser feita utilizando-se um segredo ou com um par de chaves pública/privada (JWT.IO, 2017).

JWTs podem ser enviados via URL na forma de um parâmetro no corpo de um POST ou via cabeçalho do HTTP.

Neste projeto o JWT é utilizado para a autenticação no sistema, ele contém toda a informação necessária do usuário logado. Isso evita que múltiplas consultas sejam feitas ao banco de dados para validação do usuário, necessitando de apenas uma requisição para criação do *token* que é encriptado com um segredo e enviado para o cliente. A camada cliente fica responsável por armazenar este *token* e reenviar para a API em cada requisição, o servidor então valida o *token* e recupera as informações do usuário contidas neste objeto.

3.4.5 MongoDB

MongoDB é um banco de dados *NoSQL* baseado em documentos que são salvos no formato BSON (Binary JSON), um formato similar ao formato JSON, isso significa que ele possui chaves e valores que mudam de documento para documento e sua estrutura de dados pode mudar de acordo com o tempo (MONGODB, 2017c).

Um modelo do documento mapeia os objetos do código da aplicação, fazendo com que a manipulação dos dados seja fácil e intuitiva em qualquer linguagem. Dentro da estrutura de um documento podem ser incluídos outros documentos ou *arrays* de documentos, essa estrutura composta é chamada de agregação de documentos. Documentos que compartilham as mesmas estruturas em sua definição são chamados de coleções, comparando a um banco de dados relacional, uma coleção é o equivalente a uma tabela, os documentos são as linhas, e os campos do documento são as colunas (MONGODB, 2017b).

Como resultado desta abordagem não sequencial, todos os dados necessários para um objeto geralmente estão contidos num único documento, reduzindo assim a necessidade do comando `JOIN`, ou seja, uma única operação de leitura é capaz de retornar um documento inteiro.

Outra vantagem do MongoDB é que os documentos de uma mesma coleção podem variar em sua estrutura, por exemplo, se um novo campo é adicionado num documento, os documentos salvados antes não se tornam inválidos, isto é, não há necessidade de atualizar o banco de dados inteiro para essa atualização, a partir deste momento a aplicação pode preencher esse campo de acordo com a necessidade.

Como os requisitos do sistema poderiam mudar a qualquer instante neste projeto, a justificativa para escolha do MongoDB é a possibilidade de modelar os dados de acordo com a necessidade, de maneira rápida e em tempo de desenvolvimento, sem efeitos colaterais no banco de dados. Caso utiliza-se um banco de dados relacional, as tabelas teriam que ser planejadas antes do desenvolvimento, e qualquer alteração demandaria a atualização do banco de dados, fazer isso constantemente poderia se tornar um problema. Além disso, o poder de representar os dados no banco de dados da mesma forma que na aplicação torna mais fácil o entendimento de como os documentos são salvos na base de dados.

3.4.6 GridFS

GridFS é uma ferramenta do MongoDB para salvar documentos que excedem o tamanho de documentos BSON maiores que 16mb. Em vez de salvar os dados num documento único, o GridFS divide os arquivos em partes menores chamadas de *chunks* (MONGODB, 2017a).

De acordo com a especificação do fabricante (MONGODB, 2017a) em algumas situações armazenar dados no MongoDB pode ser mais eficiente do que no sistema de arquivos do sistema operacional. É possível armazenar quantos arquivos forem necessários, sejam imagens, vídeos, arquivos de texto, etc. Com o GridFS também é possível solicitar apenas partes do arquivo sem a necessidade de carregar ele inteiro na memória. Outra vantagem é poder manter os dados atualizados e sincronizados em várias instâncias distribuídas do MongoDB.

A motivação para o uso do GridFS neste projeto se dá justamente por esse conjunto de vantagens sobre o sistema de arquivos convencional, o manuseio dos arquivos fica encapsulado pelo MongoDB, onde é possível tirar proveito dos índices, *backups*, e do sistema de *queries* do banco de dados, tudo isso sem a perda de performance. O GridFS em conjunto com o Node.js possibilita ao servidor fazer o *streaming* de arquivos direto do banco de dados, sem a necessidade de carregar o arquivo inteiro na memória antes de transmitir para o cliente.

3.4.7 Mongoose

O *Mongoose* é uma biblioteca que facilita a modelagem dos objetos que são armazenados no MongoDB. Ele disponibiliza uma solução baseada em *schemas* para modelar os dados da aplicação, abstraindo a parte de validação, *casting* de tipos, sistema de consultas e provê facilidades para lógicas de negócio (MONGOOSE, 2017).

Um *schema* é definido como um conjunto de chave/valor e seus tipos, e cada *schema* mapeia um objeto no MongoDB. A partir de um *schema* são criados os *Models* para os documentos. Um *model* é um objeto que mapeia funções para realizar *queries*, operações de adição e remoção de documentos no banco de dados, sem a necessidade do programador realizar manualmente cada uma delas.

Com o *mongoose* é fácil de manter organizado o estado dos documentos que estão sendo inseridos no MongoDB, a definição de *models* abstrai a camada de interação com

o banco de dados, dando mais foco ao programador para se dedicar à camada de negócio da aplicação.

4 PROJETO DO SISTEMA

Este capítulo apresenta em detalhes o projeto do sistema. Primeiramente serão descritas as funcionalidades necessárias na forma de *user stories*. Na segunda seção serão apresentadas as entidades do banco de dados NoSQL da aplicação. Finalmente, na última seção será documentada a API REST.

4.1 Funcionalidades Necessárias

Esta seção mostra as funcionalidades necessárias para o sistema na versão Web e *mobile*. Essas funcionalidades estão descritas na forma de *user stories*. Uma *user story* descreve do ponto de vista de um usuário uma funcionalidade desejável que agrega valor ao sistema (COHN, 2004). Geralmente elas seguem um modelo que descreve de forma narrativa o que um usuário deseja e como ele interage com uma nova funcionalidade, por exemplo:

"Como um <tipo do usuário> quero <algum objetivo> para <alguma razão>"

Neste projeto teremos apenas um tipo de usuário que é o **professor especializado**. A ideia é que futuramente o sistema comporte outros tipos de usuários.

Nas subseções seguintes são descritas as *User Stories* que estão divididas entre sistema Web e aplicativo móvel, pois as funcionalidades diferem em cada sistema.

4.1.1 User Stories do sistema Web

Nesta seção é apresentado as histórias de usuário do professor especializado que deseja utilizar o sistema Web.

Abaixo seguem as histórias de usuário do sistema web:

- Como professor quero cadastrar-me no sistema para conseguir logar.
- Como professor quero logar-me no sistema para acessar suas funcionalidades.
- Como professor quero adicionar alunos para poder registrar atividades pedagógicas associadas a eles.
- Como professor quero poder enviar fotos com comentários para registrar acontecimentos de alunos.

- Como professor quero poder enviar vídeos com comentários para registrar acontecimentos de alunos.
- Como professor quero criar documentos, com a possibilidade de inserir fotos e vídeos no campo de edição, para criar um parecer do aluno.
- Como professor quero visualizar todos os documentos de um aluno (fotos, vídeos e pareceres) em forma de *timeline* para que eu possa analisar o aluno de acordo com o tempo.
- Como professor quero visualizar um documento (fotos, vídeos ou pareceres) para que eu possa examinar um aluno.
- Como usuário quero cadastrar escolas para poder identificar a instituição de ensino de cada professor e aluno do sistema

4.1.2 User stories do aplicativo móvel

Nesta seção é apresentado as histórias de usuário do professor especializado que deseja utilizar o aplicativo móvel.

Abaixo seguem as histórias de usuário relacionadas ao aplicativo:

- Como professor quero baixar o aplicativo para poder usar o sistema no meu celular.
- Como professor quero logar-me no aplicativo para acessar suas funcionalidades.
- Como professor quero adicionar alunos para poder registrar atividades pedagógicas associadas a eles.
- Como professor quero poder capturar e enviar fotos com comentários para registrar acontecimentos de alunos.
- Como professor quero poder capturar e enviar vídeos com comentários para registrar acontecimentos de alunos.
- Como professor quero poder escolher fotos ou vídeos da galeria do dispositivo e inserir comentários para registrar acontecimentos de alunos.
- Como professor quero acessar a lista de fotos e vídeos associadas a um aluno para poder examiná-lo.

4.2 Modelagem do Banco de Dados

Esta seção detalha as entidades que compõem o sistema, nela são descritos os atributos e funcionalidades de cada coleção do banco de dados NoSQL.

4.2.1 Entidade User

Esta entidade representa um usuário do sistema, neste caso, o professor especializado. Os dados que compõem um usuário são: nome, sobrenome, tipo de usuário, e-mail, senha, número de telefone, endereço, RG e escola a qual pertence. Abaixo estão detalhados estes campos:

- **_id:** identificador único gerado automaticamente pelo MongoDB na criação de um documento.
- **name:** é um valor do tipo String que armazena o nome do professor.
- **lastName:** é um valor do tipo String que armazena o sobrenome do professor.
- **role:** é um valor do tipo String que representa o tipo do professor, neste projeto, todos são *"admin"* administradores) do sistema.
- **email:** String que armazena o e-mail do professor, para fins de login e para ser um identificador único de usuário na base de dados.
- **password:** String que armazena a senha de acesso ao sistema criada pelo usuário.
- **phone:** valor do tipo Number para armazenar o número de telefone de um usuário.
- **idRegistration:** valor do tipo Number para armazenar o RG de um usuário.
- **_school:** valor do tipo ObjectId que referencia um documento da coleção School.

4.2.2 Entidade Student

Esta entidade representa um aluno cadastrado por um usuário do sistema. É a maior entidade em quantidade de campos, nela armazenamos diversas informações sobre o aluno, entre elas, o nome, sobrenome, nome dos pais, data de nascimento, responsável,

escola, turno, turma, série, RG, número de telefone, endereço, histórico do aluno, necessidades especiais, código internacional de doenças (CID), documento com aprovação dos pais. Seguem os detalhes sobre cada campo da entidade *Student*:

- **_id:** identificador único gerado automaticamente pelo MongoDB na criação de um documento.
- **name:** valor do tipo String que armazena o nome.
- **lastName:** valor do tipo String que armazena o sobrenome.
- **motherName:** valor do tipo String que armazena o nome completo da mãe do aluno.
- **fatherName:** valor do tipo String que armazena o nome completo do pai do aluno.
- **birthDate:** valor do tipo Date que armazena a data de nascimento do aluno
- **avatar:** valor do tipo String que representa um caminho para a imagem de perfil do aluno
- **school:** String que representa a escola que o aluno está matriculado
- **classNumber:** valor do tipo Number que representa a turma que o aluno está matriculado
- **shift:** String que representa o turno (manhã, tarde, noite)
- **series:** String que armazena a série que o aluno está
- **geralRegister:** valor do tipo Number que armazena o RG do aluno
- **phoneNumber:** valor do tipo Number que armazena o número de telefone para contato com a família
- **responsible:** String que representa o responsável por esse aluno
- **relationship:** String descrevendo a relação do aluno com o responsável
- **address:** String que armazena o endereço onde o aluno reside
- **cid:** Number que representa o código internacional de doenças relacionado a este aluno

- **historic:** String que armazena o histórico de atendimentos deste aluno
- **specialNeeds:** Array de Strings que armazena as possíveis necessidades especiais do aluno
- **docParentsAproval:** String do caminho para o documento de aprovação dos pais para uso de imagem e cadastro de informações
- **termOfUse:** Boolean que armazena a informação de se o professor especializado leu e concordou com os termos e uso de imagem do aluno
- **_createdBy:** String ou ObjectId de referência para a coleção User, representando o usuário que criou o aluno no sistema
- **_schoolId:** valor do tipo String ou ObjectId que referencia um documento da coleção School.

4.2.3 Entidade Judgement

Esta entidade representa um parecer do professor para determinado aluno. Um parecer basicamente tem uma referência para um documento da coleção *Student* e para o usuário que está o criando, em seus outros campos armazena o texto do conteúdo do parecer, data de criação, título opcional e o tipo de documento. Segue abaixo as especificações destes campos.

- **_id:** identificador único gerado automaticamente pelo MongoDB na criação de um documento.
- **_createdBy:** valor do tipo String ou ObjectId que armazena uma referência para a coleção User, que representa o usuário que criou o parecer.
- **_studentId:** valor do tipo String ou ObjectId que armazena uma referência para a coleção Student, que tem o papel de associar o parecer a um aluno da base de dados.
- **title:** String que armazena o título do parecer
- **text:** String que armazena o corpo do texto do parecer
- **date:** String que armazena a data de criação

- **type:** String que armazena o tipo de documento, neste caso, será sempre "judgement"(parecer). Este campo foi introduzido para futuramente facilitar a diferenciação de documentos no *Front-End* do projeto, pode ser omitido pelo programador na hora da criação, o *Back-End* preencherá automaticamente este atributo.

4.2.4 Entidade File

Esta entidade representa um arquivo de imagem ou vídeo dentro da base de dados. Como mencionado no capítulo anterior, especificamente na seção **3.3.6**, os arquivos são armazenados numa coleção especial do MongoDB, que divide um documento maior em diversos outros documentos menores. Essa coleção no MongoDB tecnicamente se chama `fs.files`, porém para o propósito deste trabalho renomearemos apenas para ”*File*”. A entidade *Files* armazena o arquivo em si e também alguns metadados associados ao arquivo, esses metadados são definidos no *Front-End* da aplicação e não possuem estrutura, ou tipo definido, dos atributos por causa da flexibilidade do MongoDB. Com o intuito de definir uma estrutura padrão, utilizaremos como metadados as seguintes informações: referência para o usuário que está salvando o arquivo, comentário sobre a foto ou vídeo, data do acontecimento, título para o arquivo e também armazenaremos uma referência para a coleção *Student* na qual o arquivo está associado.

Abaixo segue um modelo de representação dessa estrutura que o GridFS gera:

- **_id:** identificador único gerado automaticamente pelo MongoDB na criação de um documento.

- **length:** Number preenchido pelo GridFS ao armazenar um documento,

- **chunkSize:** Number preenchido pelo GridFS ao armazenar um documento.

- **uploadDate:** Date preenchido pelo GridFS ao armazenar um documento

- **md5:** Number preenchido pelo GridFS ao armazenar um documento,

- **filename:** String que armazena o nome do arquivo a ser salvo na base de dados.

- **contentType:** String que armazena se o arquivo a ser salvo é um vídeo, imagem ou outros formatos.

- **metadata:** Objeto definido pelo programador para armazenar os metadados associados ao documento de imagem ou vídeo. No MongoDB os metadados são armazenados em uma coleção separada da coleção file. Para representar esta coleção definimos os seguintes metadados:

  _createdBy: valor do tipo String ou ObjectId que armazena uma referência para a coleção User, que representa o usuário que adicionou o arquivo de foto ou vídeo

  _studentId: valor do tipo String ou ObjectId que armazena uma referência para a coleção Student, associando o arquivo a ser adicionado ao aluno.

  title: valor to tipo String que armazena um título para o arquivo.

  comment: valor do tipo String que armazena um comentário ao arquivo.

  date: valor do tipo Date que armazena a data do acontecimento que contém no arquivo.

4.2.5 Entidade School

Esta entidade representa uma escola ou instituição de ensino na qual os professores especializados e alunos estão associados. Abaixo segue a descrição dos campos na base de dados:

- **_id:** identificador único gerado automaticamente pelo MongoDB na criação de um documento.
- **name:** valor do tipo String que armazena o nome da escola
- **mapCoordinates:** valor do tipo Array de Numbers com duas posições que representam as coordenadas (longitude, latitude) da escola.
- **cep:** valor do tipo Number que armazena o CEP da escola.
- **adress:** valor do tipo String que armazena o endereço.
- **district:** valor do tipo Sting que armazena o município.
- **city:** String que armazena a cidade em que se localiza a instituição de ensino
- **state:** String que armazena o estado

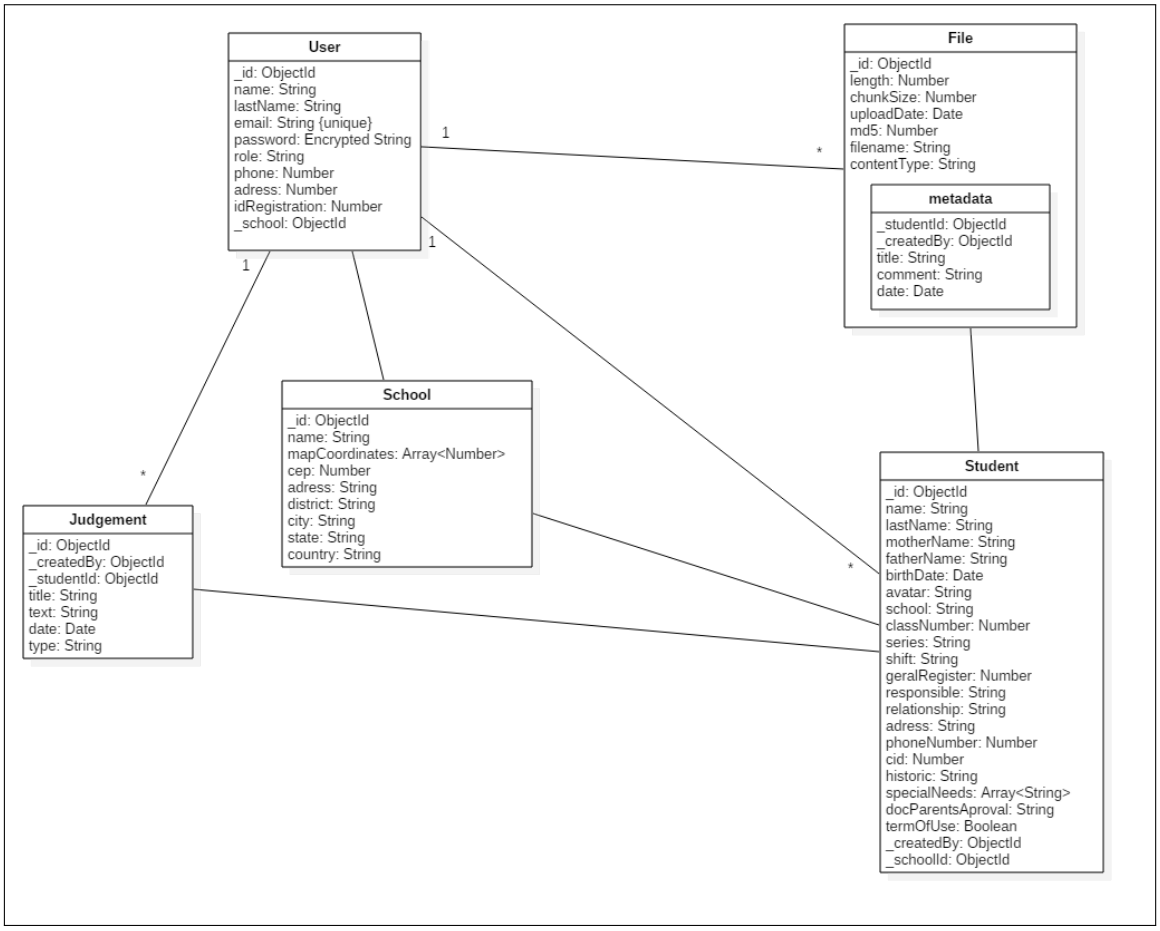
- **country:** String que armazena o país

4.2.6 Diagrama de Entidades

Esta seção descreve como as entidades estão conectadas utilizando um diagrama de classes *UML*.

Cada classe representa uma coleção de documentos do banco de dados NoSQL, dentro delas estão detalhados os campos e seus tipos. Para representar a agregação de documentos, ou seja, quando um documento é um valor dentro de outro documento, utilizamos a notação de uma classe dentro de outra classe *UML*. Por exemplo, na Figura 4.1 pode-se observar que a coleção *metadata* é apresentada embutida na coleção *File*.

Figura 4.1: Diagrama das entidades da base de dados NoSQL



Fonte: Elaborada pelo autor

4.3 API REST

Nesta seção é descrita a interface de acesso aos recursos do sistema. Nas suas subseções a API REST desenvolvida está organizada na forma de tabelas, cada uma delas contendo informações sobre as URLs de acesso, parâmetros para as requisições, métodos HTTP e respostas às requisições.

Como por definição uma API REST não deve manter estados, o cliente fica responsável por controlar a sessão do usuário autenticado. Neste projeto utilizamos a tecnologia de *JSON Web Tokens* apresentada no capítulo 2.3.4, portanto o cliente fica encarregado de enviar, dentro do cabeçalho HTTP, o *token* de acesso ao sistema. Este *token* é enviado da seguinte forma:

```
Authorization:  Bearer <token>
```

Todas as rotas protegidas do sistema irão procurar o token no cabeçalho, decodificar, validar e recuperar as informações do usuário logado, sem a necessidade de realizar uma consulta adicional na base de dados. Caso o token seja inválido ou não esteja presente no cabeçalho, um código de erro é retornado ao cliente informando que ele não está autorizado para acessar esse recurso. Nas tabelas esse parâmetro de autorização não está descrito, apenas estão descritos os parâmetros internos necessários para determinada rota da API.

A API deste trabalho está hospedada nos servidores do heroku [1] e tem o seguinte formato de acesso: *https://sir-edu.heroku.com/api/*. Por exemplo, para acessar os recursos da entidade *students* as requisições de criação, deleção, modificação e leitura devem ser destinadas à seguinte URL: *https://sir-edu.heroku.com/api/students/*.

[1] <http://www.heroku.com/>

4.3.1 User

Tabela 4.1: Tabela de operações para o recurso user

Método	*URI*	*Parâmetros*	*Funcionalidade*	*Resposta*
POST	/user/register	JSON no formato da entidade user	Adiciona um novo usuário	Objeto JSON no formato da entidade user
POST	/user/authenticate	JSON contendo email e senha de um usuário	Autenticação e criação do token de acesso ao sistema	Token de acesso ao sistema
GET	/user/{id}	id do usuário	Retorna um usuário com o id informado	Objeto JSON no formato da entidade user
GET	/user	Nenhum	Retorna todos os usuários da base de dados	JSON contendo un Array de Objetos no formato da entidade user
PUT	/user/{id}	id do usuário e JSON no formato da entidade user	Altera os dados de um usuário	Objeto JSON no formato da entidade user
DELETE	/user/{id}	id do usuário	Remove um usuário da base de dados	Objeto JSON no formato da entidade user

Fonte: Elaborada pelo autor

4.3.2 Students

Tabela 4.2: Tabela de operações para o recurso students

Método	*URI*	*Parâmetros*	*Funcionalidade*	*Resposta*
POST	/students	JSON no formato da entidade student	Adiciona um novo aluno	Objeto JSON no formato da entidade student
GET	/students/{id}	id do aluno	Retorna um aluno com o id informado	Objeto JSON no formato da entidade aluno
GET	/students	Nenhum	Retorna todos os alunos que o usuário logado tem acesso	JSON contendo un Array de Objetos no formato da entidade student
PUT	/students/{id}	id do aluno e JSON no formato da entidade aluno	Altera os dados de um usuário	Objeto JSON no formato da entidade aluno
DELETE	/students/{id}	id do aluno	Remove um aluno da base de dados	Objeto JSON no formato da entidade aluno

Fonte: Elaborada pelo autor

4.3.3 Files

Tabela 4.3: Tabela de operações para o recurso files

Método	*URI*	*Parâmetros*	*Funcionalidade*	*Resposta*
POST	/files	*FormData* contendo o upload do arquivo e dados na forma da entidade files	Adiciona um arquivo para a um aluno	Objeto JSON no formato da entidade files
GET	/files/{id}	id do arquivo	Retorna um arquivo da base de dados	Download do arquivo via streaming de dados
GET	/files/:studentId	id do aluno relacionado aos arquivos	Recupera todos os arquivos associados a um aluno	JSON contendo un Array de Objetos no formato da entidade files
DELETE	/files/{id}	id do arquivo	Remove um arquivo da base de dados	Objeto JSON no formato da entidade files

Fonte: Elaborada pelo autor

4.3.4 Judgement

Tabela 4.4: Tabela de operações para o recurso judgement

Método	*URI*	*Parâmetros*	*Funcionalidade*	*Resposta*
POST	/judgement	JSON no formato da entidade judgement	Adiciona um parecer do professor logado para um aluno	Objeto JSON no formato da entidade judgement
GET	/judgement/{id}	id do parecer	Retorna um parecer com o id informado	Objeto JSON no formato da entidade judgement
GET	/judgement/:studentId	id do aluno relacionado ao parecer	Recupera todos os pareceres associados a um aluno	JSON contendo un Array de Objetos no formato da entidade judgement
PUT	/judgement/{id}	id do parecer e JSON no formato da entidade judgement	Altera os dados de um parecer	Objeto JSON no formato da entidade judgement
DELETE	/judgement/{id}	id do parecer	Remove um parecer da base de dados	Objeto JSON no formato da entidade judgement

Fonte: Elaborada pelo autor

4.3.5 School

Tabela 4.5: Tabela de operações para o recurso school

Método	*URI*	*Parâmetros*	*Funcionalidade*	*Resposta*
POST	/schools	JSON no formato da entidade school	Adiciona uma escola na base de dados	Objeto JSON no formato da entidade school
GET	/schools/{id}	id do parecer	Retorna uma escola com o id informado	Objeto JSON no formato da entidade school
GET	/schools	Nenhum	Recupera todas as escolas da base de dados	JSON contendo un Array de Objetos no formato da entidade school
PUT	/schools/{id}	id da escola e JSON no formato da entidade school	Altera os dados de um parecer	Objeto JSON no formato da entidade school
DELETE	/schools/{id}	id da escola	Remove uma escola da base de dados	Objeto JSON no formato da entidade school

Fonte: Elaborada pelo autor

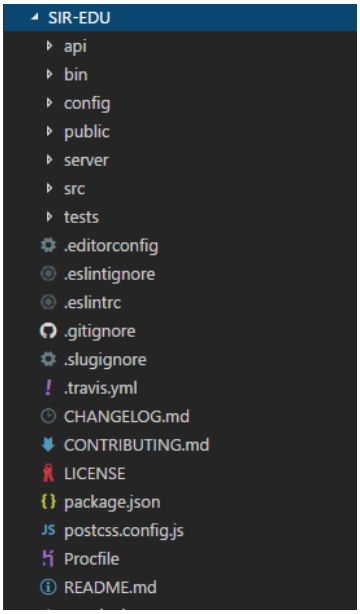
5 IMPLEMENTAÇÃO DO PROJETO

Este capítulo tem o intuito de descrever de maneira prática o desenvolvimento do projeto e o emprego das tecnologias mencionadas no capítulo 4. São mostrados exemplos de códigos, organizações de arquivos e configurações necessárias para o funcionamento do sistema como um todo. Na primeira seção é exibido a estrutura e desenvolvimento da API na camada do servidor da aplicação. Na segunda seção é demonstrada a estrutura do sistema web e detalhes de sua implementação. A última seção descreve detalhes relativos ao projeto do aplicativo mobile.

5.1 Back End do Sistema

Nesta seção são descritos os componentes necessários para o funcionamento da API e servidor do sistema. O servidor da aplicação é o responsável por disponibilizar a API e servir o conteúdo estático como imagens, ícones, arquivos JavaScript, CSS, etc. A Figura 5.1 ilustra a organização dessas pastas e arquivos:

Figura 5.1: Estrutura de arquivos no servidor



Fonte: Elaborada pelo autor

A pasta *api* contém os arquivos necessários para criação da API do sistema. A pasta *src* armazena os arquivos estáticos relativos ao sistema web, o servidor tem a função de gerenciar as dependências do *Front-End*, empacotar todos esses arquivos e servi-los

para o navegador de internet do usuário. Os demais arquivos presentes na pasta raiz são necessários para configuração do ambiente de execução, dependências de pacotes, integração com o *GitHub*, descrição do projeto, etc. As subseções seguintes abordam os detalhes da implementação e estrutura do código da API.

5.1.1 Configuração com npm

O arquivo `package.json` localizado na pasta raiz da Figura 5.1 contém as configurações necessárias para resolução das dependências de pacotes de um projeto que utiliza o *npm*. Este arquivo contém informações sobre o versionamento de cada pacote, e também outros diversos *scripts* com comandos para inicialização do ambiente de execução na máquina local, *deploy* do código no servidor, verificação de erros de JavaScript e HTML, execução de testes unitários, etc. Neste projeto o arquivo `package.json` ficou relativamente extenso, pois o *Back-End* e o *Front-End* da aplicação compartilham o mesmo arquivo de configuração, dentre as dependências, as principais são: Express, React, Mongoose, JWT (JSON Web Token), MongoDB driver, GridFS e Webpack.

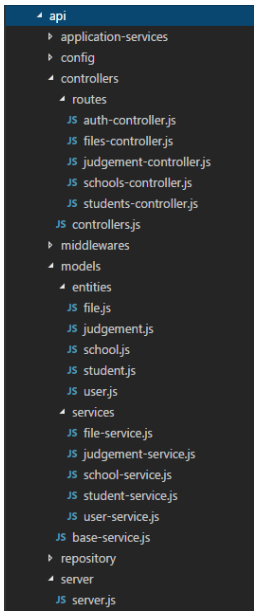
Com o *npm* torna-se fácil o compartilhamento do projeto, pois as mesmas versões de pacotes utilizadas na configuração inicial são mantidas para as demais instalações do sistema. Para instalar tais dependências é necessário executar o comando `npm install` no console a partir do diretório raiz da aplicação, após baixar todas essas dependências, um novo ambiente de desenvolvimento é facilmente configurado com o comando `npm run dev`, que executará as tarefas necessárias para inicialização do servidor.

5.1.2 Estrutura da API do sistema

A Figura 5.2 ilustra a organização de pastas e arquivos que compõem a API REST do sistema.

A API está organizada entre *controllers*, *models* e *services*. Dentro da pasta *controllers* são definidas as *routes* da API, onde são processadas as requisições das aplicações cliente. A pasta *models* armazena a modelagem das entidades do banco de dados do sistema e também os serviços com comandos para manipulação desses modelos. Basicamente os *controllers* são definidos com o *framework Express*, os *models* são definidos com o Mongoose e os serviços são os métodos de acesso aos documentos do MongoDB,

Figura 5.2: Estrutura de arquivos da API



Fonte: Elaborada pelo autor

que também usufruem do *framework Mongoose* para a abstração dos comandos de manipulação do banco de dados. Os demais arquivos são importantes para configuração da API, entre eles, o arquivo `server.js` é o mais importante de todos, pois nele são importadas todas as dependências necessárias para seu funcionamento, além disso, nele é definido o fluxo das requisições, a porta que o servidor irá executar e o armazenamento e recuperação dos arquivos estáticos da camada de *Front-End*.

5.1.3 Autenticação

A estratégia de autenticação do projeto foi desenvolvida utilizando *JSON Web Tokens*. O processo de autenticação exige que o usuário informe seus dados de e-mail e senha, e então o *back-end* processa a requisição e verifica se o usuário está apto a utilizar o sistema.

Na Figura 5.3 apresenta-se o trecho de código que processa a requisição de autenticação.

Observa-se no código que o ao chegar a requisição de autenticação, o controle é passado para a função `authentication` que recebe os dados de um usuário, verifica se ele existe na base de dados, e caso exista, cria-se o *token* de acesso na função `_create_token` com a biblioteca *jwt*, que gera JWT encriptado contendo os dados do

usuário e uma data de expiração. Esse *token* é retornado como reposta para a aplicação cliente, que deve armazená-lo e enviá-lo nas futuras requisições à API.

Figura 5.3: Trecho de código de autenticação

```
_createToken(user) {
    return jwt.sign({email: user.email, role: user.role, _id: user._id}, appConfig.secret, {
      expiresIn: 1440*60
    });
}

authenticate(user) {
    return this.userService.findOne({ email : user.email })
        .then((userFound) => {
            if(!userFound) {
                throw new Error(404);
            }
            return userFound.comparePassword(user.password)
                .then((isMatch) => {
                    if(isMatch){
                        let token = this._createToken(userFound);
                        return {
                            token: token,
                            user: userFound
                        };
                    }
                    else{
                        throw new Error('O usuário não foi encontrado ou a senha informada esta incorreta.
                    }
                })
                .catch((err) => { throw err; });
        })
        .catch((err) => {
            throw err;
        });
}
```

Fonte: Elaborada pelo autor

5.1.4 Definição das entidades com *schemas* do Mongoose

O MongoDB não possui uma estrutura fixa para definição das coleções de documentos, pois os valores podem variar para cada chave-valor. O Mongoose é um *framework* que ajuda a criar e manter essas coleções se baseando em *schemas*. Um *schema* define uma estrutura para coleções, contendo chaves e valores tipados, facilitando o controle sobre os dados armazenados no MongoDB. A partir de cada *schema* é criado um objeto chamado *model*, que possui métodos que abstraem a comunicação e validação dos dados com o MongoDB. Esses métodos permitem inserir, modificar, deletar e pesquisar dados no MongoDB sem o programador acessar diretamente o banco de dados, assim a consistência é garantida e há menos espaço para erros nessa camada.

Na Figura 5.4 tem-se um exemplo de *schema* e *model* da entidade *student* para o protótipo deste projeto. Pode-se observar que temos uma coleção com chaves e valores, onde cada valor possui um tipo definido para ser armazenado no MongoDB. Depois de definir o *schema*, um *model* é criado e exportado para ser usado em outros componentes do sistema.

Figura 5.4: Schema da entidade student

```
let mongoose = require('mongoose'),
 Schema = mongoose.Schema;

let StudentSchema = new Schema({
  name: { type: String },
  lastName: { type: String },
  fatherName: { type: String },
  motherName: { type: String },
  birthDate: { type: Date},
  avatar: { path: String, mimeType: String },
  school: { type: String },
  classNumber: { type: Number },
  series: { type: String },
  shift: { type: String },
  geralRegister: { type: Number },
  phoneNumber: { type: Number },
  responsible: { type: String},
  relationship: { type: String},
  address: { type: String},
  cid: { type: Number},
  historic: { type: String},
  specialNeeds: [{ type: String }],
  docParentsAproval: { path: String, mimeType: String },
  termOfUse: { type: Boolean },
  _createdBy: { type: String, ref: 'User' }, //Link to teacher model
  _schoolId: { type: String, ref: 'School' } //Link to school model
 });

// set up a mongoose model
const Student = mongoose.model('Student', StudentSchema);
```

Fonte: Elaborada pelo autor

5.1.5 Definição das rotas da API com o Express

Como visto na Figura 5.2, as rotas do Express estão definidas dentro da pasta `routes` localizada nos `controllers` da aplicação. Cada rota segue o padrão de nomeação de arquivos da forma <*entidade-controller.js*>. Neste projeto cada conjunto de rotas do Express é responsável por processar requisições e manipular os recursos associados a uma entidade. Uma rota do Express é composta por um método derivado do HTTP, seguido de uma URL que define um *endpoint* no qual as requisições são destinadas. Para cada combinação de método e URL que constituem uma rota, um *call-back* chamado de *middleware* é executado para lidar com as requisições da camada cliente, processando os dados envidados e devolvendo a resposta apropriada.

O *framework Express* foi construído para facilitar a criação de aplicações REST, todas as definições de rotas neste projeto seguem o padrão REST predefinido pelo *framework*.

Na Figura 5.5 tem-se um exemplo da criação de uma rota para retornar um aluno da base de dados. Essa rota responde às requisições destinadas à URL no formato `/students/:id` que utilizarem o método *GET* do HTTP. O controle da requisição é passado para o *middleware* que recupera o *:id* do aluno presente na URL, e também o usuário decodificado do *token* enviado no corpo da requisição *req*. Após recuperar esses dados, é processada

a pesquisa do aluno com os dados do contexto, retornando o objeto JSON do aluno, caso ele exista e o usuário tenha controle sobre o aluno com o id informado, senão é retornado um código de erro.

Figura 5.5: Exemplo de rota do Express

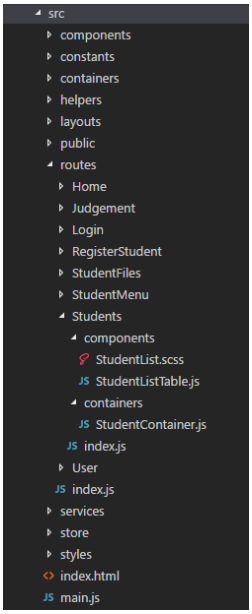
```
router.get('/students/:id', function(req, res, next) {
    studentService.find({_createdBy: req.user._id, _id: req.params.id})
        .then((data) => {
            res.json(data);
        })
        .catch((err) => {
            next(err);
        })
});
```

Fonte: Elaborada pelo autor

5.2 Sistema web

Este protótipo utiliza uma arquitetura baseada em componentes, por causa disso, a estrutura de arquivos está organizada de forma a explorar a modularização do código. A Figura 5.6 ilustra essa estrutura da aplicação web desenvolvida com ReactJS.

Figura 5.6: Estrutura de arquivos do sistema web



Fonte: Elaborada pelo autor

Essa estrutura está basicamente organizada em *components*, *routes*, *stores* e *servi-*

ces. A pasta *components*, que se encontra na raiz do projeto, armazena os componentes de uso comum na aplicação, ou seja, os componentes que são utilizados por diferentes páginas. A Figura 5.7 ilustra um exemplo de um componente em React.

Figura 5.7: Exemplo do componente *DialogModal* em *React*

```
import { FlatButton, Dialog } from 'material-ui'
import React from 'react'

class DialogModal extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        const actions = [
            <FlatButton
                label="Ok"
                primary={true}
                onTouchTap={this.props.onClose}
            />
        ]

        return (
            <Dialog
                actions={actions}
                modal={true}
                open={this.props.open}
            >
                {this.props.children}
            </Dialog>
        );
    }
}

export default DialogModal;
```

Fonte: Elaborada pelo autor

A pasta *routes* armazena as rotas internas da aplicação, essas rotas refletem o que o usuário vê na URL do navegador de internet, e é através delas que se controla o fluxo da aplicação e renderização dos componentes. Cada rota possui seus componentes internos e arquivos de estilos CSS, separados entre *Components* e *Containers*. Um *container* nada mais é do que um componente especial que engloba componentes menores, ele é responsável por se conectar a uma *store* e repassar os dados do estado da aplicação para os componentes filhos. Para definição de uma rota é necessário declarar uma URL de acesso e um componente para renderização, por exemplo, uma rota para listagem de alunos seria:

Figura 5.8: Declaração de uma rota

```
1 import StudentContainer from './containers/StudentContainer'
2
3 export default {
4     path: 'alunos',
5     component: StudentContainer
6 }});
7
```

Neste caso, quando o usuário for redirecionado para a rota */alunos*, o componente *StudentContainer* será *renderizado* dinamicamente na tela do navegador.

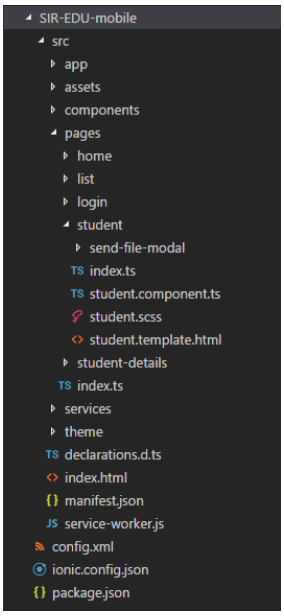
A pasta `store` armazena as *stores* da aplicação, nela são armazenados os dados relativos a uma entidade do banco de dados. Numa *store* são definidas *actions* e *handlers* para essas ações. Uma ação geralmente performa alguma requisição AJAX e o *handler* dessa *action* armazena o resultado na sua *store*, os componentes conectados a uma *store* esperam por essas mudanças nos dados para poderem se atualizar.

A pasta `services` contém os serviços do sistema, um serviço é uma classe que contém configurações e métodos para efetuar requisições AJAX. Os serviços são organizados desta forma para poderem ser importados em diferentes partes da aplicação.

5.3 Aplicativo Mobile

Pode-se considerar que o aplicativo mobile é um projeto à parte, pois ele possui seu próprio arquivo de configuração de pacotes `package.json` e não compartilha dependências com o protótipo web ou com a API. A Figura 5.9 ilustra a estrutura de arquivos do protótipo do aplicativo.

Figura 5.9: Estrutura de arquivos do sistema web



Fonte: Elaborada pelo autor

A principal dependência de pacotes do protótipo do aplicativo é o *framework Ionic*, o *Apache Cordova* também está incluído no *framework*. O Ionic disponibiliza uma série de *scripts* que auxiliam no processo de desenvolvimento, dentre eles há opções para compilar a aplicação para um dispositivo, gerar um app para testar em tempo de de-

senvolvimento e também emular o aplicativo no navegador de internet do computador. Após baixar as dependências com o comando `npm install`, basta utilizar o comando `ionic run <dispositivo destino>` para instalar e começar a testar o protótipo num dispositivo Android, IOS ou Windows Phone.

Assim como no sistema web, o aplicativo mobile se baseia na arquitetura de componentes, porém utilizamos aqui os componentes provenientes do *framework Ionic* para construir as telas.

De acordo com estrutura ilustrada na Figura 5.9, a pasta `app` contém arquivos importantes para a configuração inicial do aplicativo, neles são importados os arquivos HTML, JavaScript e CSS do restante da aplicação. O diretório `pages` armazena as páginas do sistema, cada página contém pelo menos um arquivo HTML, JavaScript e CSS em sua construção. A pasta `services` armazena os serviços com as configurações necessárias para gerar as requisições AJAX no aplicativo.

A Figura 5.10 ilustra a utilização dos componentes do Ionic para a construção da página de login da aplicação. Pode-se observar que os componentes iniciam com o prefixo `"ion-"` seguido do seu identificador.

Figura 5.10: Código HTML da tela de login da aplicação utilizando Ionic

```
<ion-header>
  <ion-navbar>
    <ion-title>Login</ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
    <form (ngSubmit)="onSubmit(LoginForm)" #LoginForm="ngForm">
        <ion-list>
            <ion-item>
                <ion-label floating>Username</ion-label>
                <ion-input type="email"
                    [(ngModel)]="loginForm.email"
                    name="email"
                    #name="ngModel"
                    required>
                </ion-input>
            </ion-item>
            <ion-item>
                <ion-label floating>Password</ion-label>
                <ion-input type="password"
                    [(ngModel)]="loginForm.password"
                    name="password"
                    #name="ngModel"
                    required>
                </ion-input>
            </ion-item>
        </ion-list>
        <button type="submit" ion-button full large color="secondary">Entrar</button>
    </form>
</ion-content>
```

Fonte: Elaborada pelo autor

6 INTERFACES DE NAVEGAÇÃO

Este capítulo apresenta as telas e funcionalidades do projeto desenvolvido. Também descreve uma avaliação de uso do sistema, identificando melhorias. As seções a seguir estão organizadas em interface do sistema web, interface do aplicativo mobile e, por fim, avaliação dos sistemas.

6.1 Sistema web

Nesta seção são apresentados os resultados obtidos na forma de imagens das telas do sistema web, dando exemplos do fluxo das atividades de um professor ao utilizar o sistema.

6.1.1 Tela de login

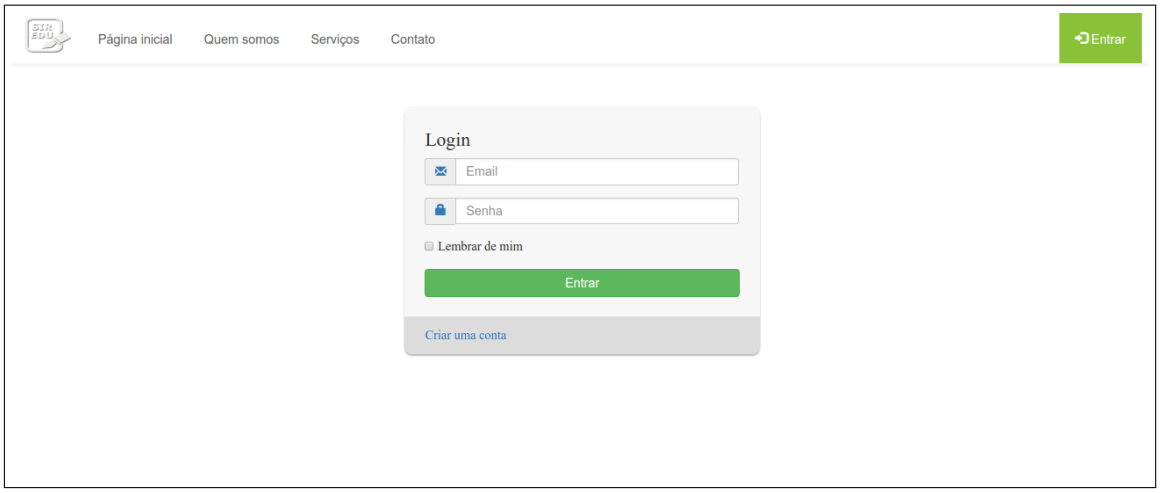
A tela de login é exibida quando o usuário pressiona o botão de ”Entrar” localizado no cabeçalho da tela do sistema. Para poder acessar o sistema o professor deverá se autenticar inserindo seu e-mail e senha previamente cadastrados.

Caso o professor ainda não tenha cadastro, este poderá ser feito pressionando o link "Criar uma conta", exibido na Figura 6.1. Inicialmente o cadastro está aberto para qualquer professor que deseja utilizar este protótipo. Todos se cadastram como administradores, portanto possuem todas as permissões para gerenciamento de alunos.

6.1.2 Tela de cadastro de usuário

Caso seja a primeira vez que o professor acessa o sistema, é necessário criar sua conta de acesso. Na Figura 6.2 estão listadas as informações necessárias para o preenchimento do cadastro, após a inserção dessas informações, pressionando-se o botão "Cadastrar", uma mensagem de sucesso ou erro será mostrada ao usuário.

Figura 6.1: Tela de login



Fonte: Elaborada pelo autor

Figura 6.2: Tela de cadastro de usuário

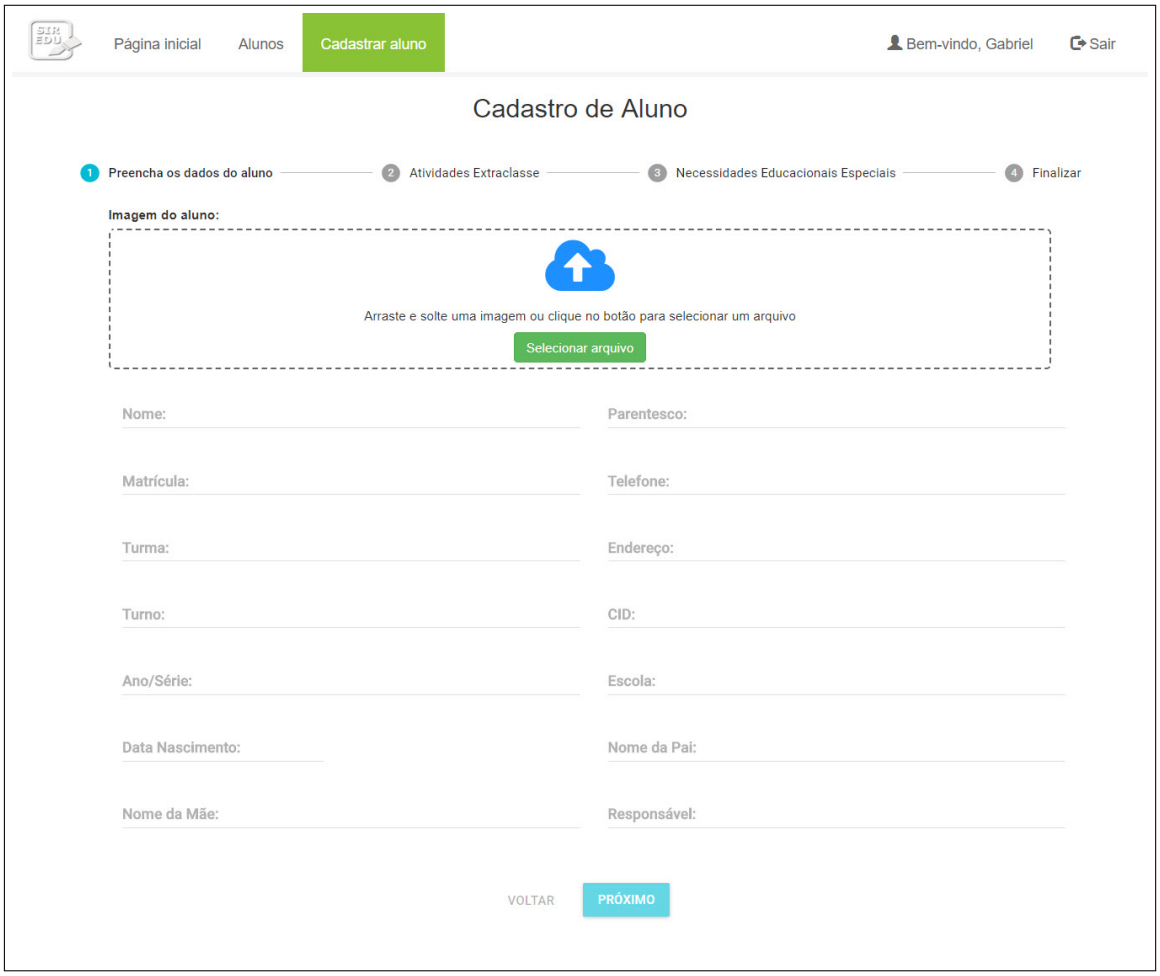
Fonte: Elaborada pelo autor

6.1.3 Cadastro de aluno

Para acessar a tela de cadastro de alunos é necessário pressionar a aba ”Cadastrar aluno” presente no cabeçalho da aplicação, ilustrado na Figura 6.3. O cadastro de um aluno foi dividido em quatro etapas, para melhor disposição do conteúdo e redução do tamanho da página, tornando a navegação mais fluída. Com isso, tem-se as categorias para inserir dados gerais do aluno, informações sobre atividades extraclasse, descrição

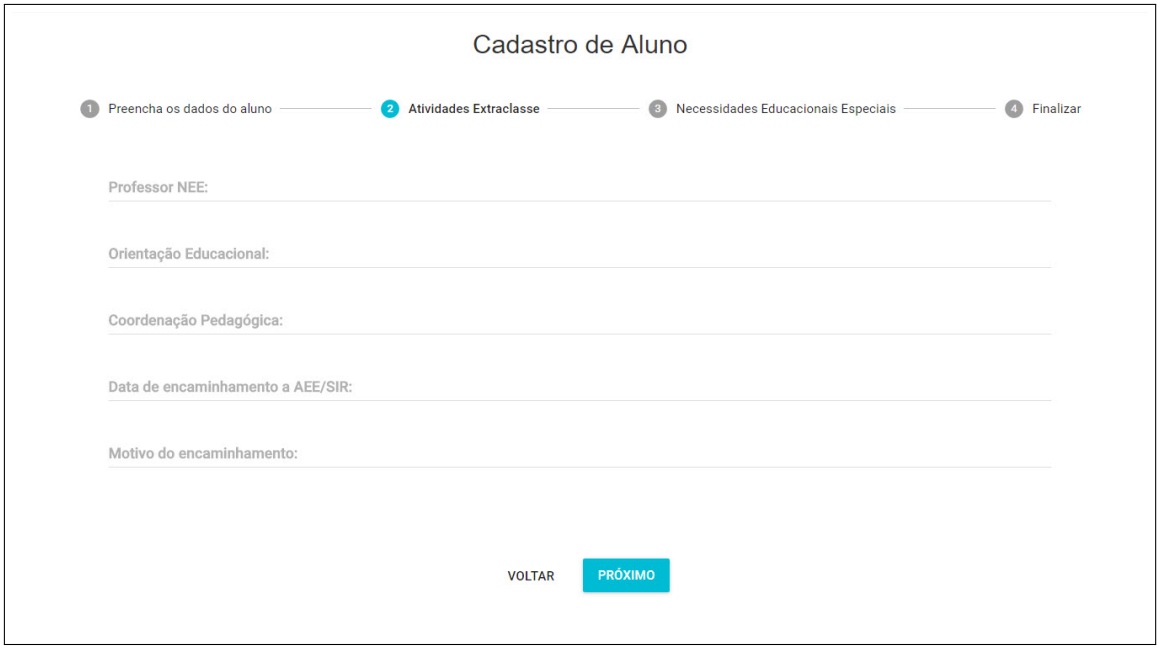
das necessidades especiais e, por fim, a etapa de aceite dos termos de uso e imagem de um aluno, e também a inserção do documento assinado pelos pais dando permissão ao professor para realizar atividades com seu filho. Para navegar entre as etapas do cadastro, o usuário deve utilizar os botões de “Voltar e “Próximo. Na última etapa está presente o botão ”Cadastrar” para realizar a ação de cadastrar um aluno.

Figura 6.3: Etapa 1: Dados do aluno



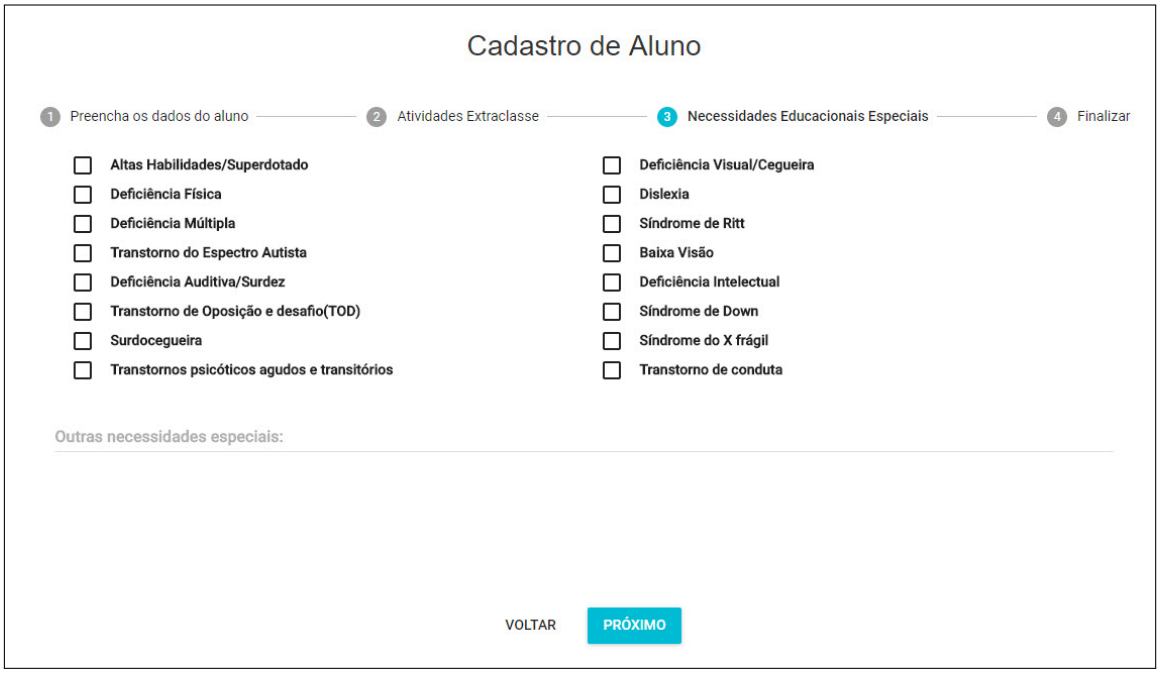
Fonte: Elaborada pelo autor

Figura 6.4: Etapa 2: Atividades extraclasse



Fonte: Elaborada pelo autor

Figura 6.5: Etapa 3: Necessidades educacionais especiais



Fonte: Elaborada pelo autor

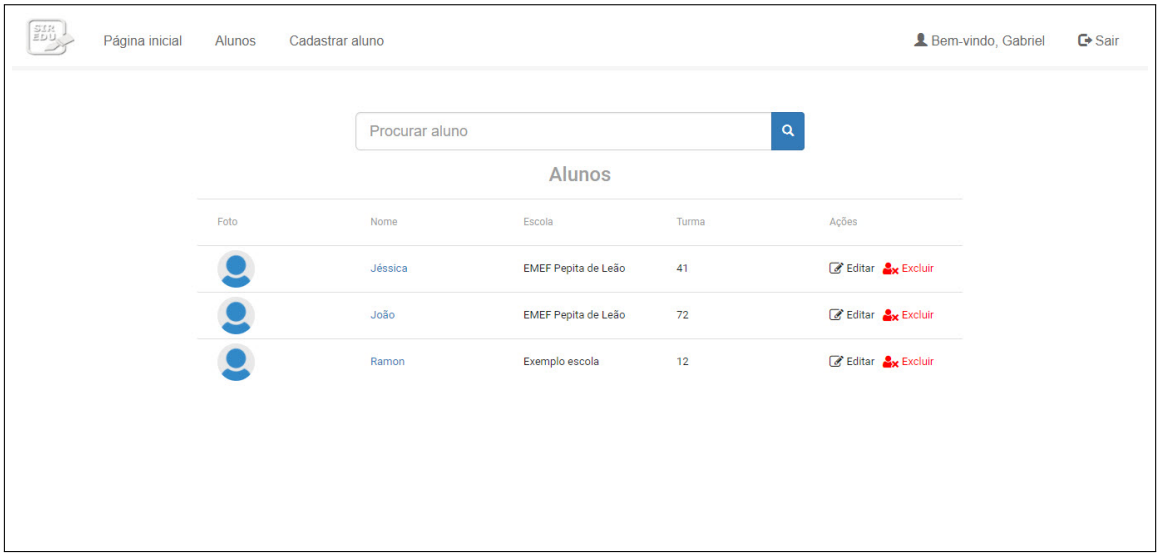
Figura 6.6: Etapa 4: Finalização cadastro

Fonte: Elaborada pelo autor

6.1.4 Tela da lista de alunos

A tela da lista de alunos é mostrada assim que o professor se loga no sistema. No topo desta página há um campo de busca para pesquisar um aluno da lista, logo abaixo tem-se uma tabela listando cada aluno que o professor cadastrou e possui acesso. Em cada linha da tabela são apresentadas as seguintes informações do aluno: foto de perfil, nome, escola, turma e um campo de ações com dois botões, um para editar o aluno e outro para excluí-lo da base de dados. A Figura 6.7 ilustra uma lista com três alunos fictícios.

Figura 6.7: Lista de alunos



Fonte: Elaborada pelo autor

6.1.5 Menu do aluno

Ao clicar no nome de um aluno da lista, o usuário será redirecionado para a página de menu de registro deste aluno. Nessa página, tem-se um cabeçalho com informações relativas ao aluno sendo gerenciado, ela ficará fixa no topo em todas as demais páginas filhas. Logo abaixo tem-se os ícones do menu contendo cinco opções de registro, porém para os propósitos deste protótipo, foram implementadas apenas as funcionalidades de registro de fotos/vídeos, e criação de pareceres. As demais opções como registro de atendimento, adequação curricular e P.D.I (Plano de Desenvolvimento Individual) são apenas ilustrativas para compor a interface de usuário definida durante o projeto do sistema.

Figura 6.8: Tela do menu do aluno

Fonte: Elaborada pelo autor

6.1.6 Tela de envio de arquivos

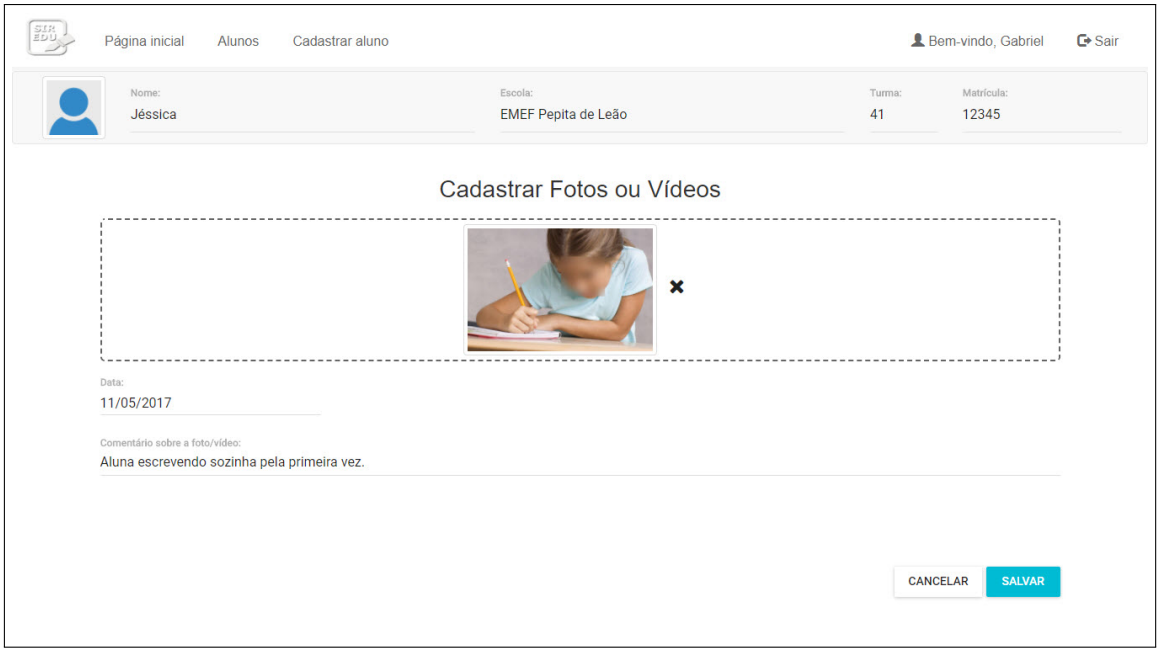
Ao clicar no menu de registro de ”Fotos/Vídeos” ilustrado na Figura 6.8, o usuário é redirecionado para a subpágina de cadastro de arquivos representada na Figura 6.9. No topo dessa página tem-se a opção de enviar arquivos acessando as pastas do computador clicando no botão “Selecionar arquivo ou há a opção de realizar o *drag and drop*(arrastar e soltar) de um arquivo diretamente do computador para o navegador, funcionalidade encontrada em diversas páginas web modernas. Logo abaixo podem ser vistos dois campos, um para inserir a data do acontecimento, e o outro para deixar um comentário relativo ao arquivo. No final da imagem há dois botões, um para cancelar a operação e voltar para o menu de registro, e o outro para salvar o arquivo na base de dados. O botão de salvar aparece desabilitado para o usuário pois não há informações preenchidas no momento.

A Figura 6.10 apresenta um exemplo de cadastro com dados fictícios para uma imagem de um aluno. Observa-se a imagem de uma menina realizando a atividade de escrita pela primeira vez, e logo ao lado temos um botão de remoção, que serve para remover a imagem caso o usuário tenha o desejo de escolher uma outra. No campo de data tem-se a data do acontecimento e no campo de comentário tem-se uma breve descrição do contexto que foi tirada a foto.

Figura 6.9: Tela de envio de arquivos

Fonte: Elaborada pelo autor

Figura 6.10: Exemplo de cadastro de uma foto



Fonte: Elaborada pelo autor

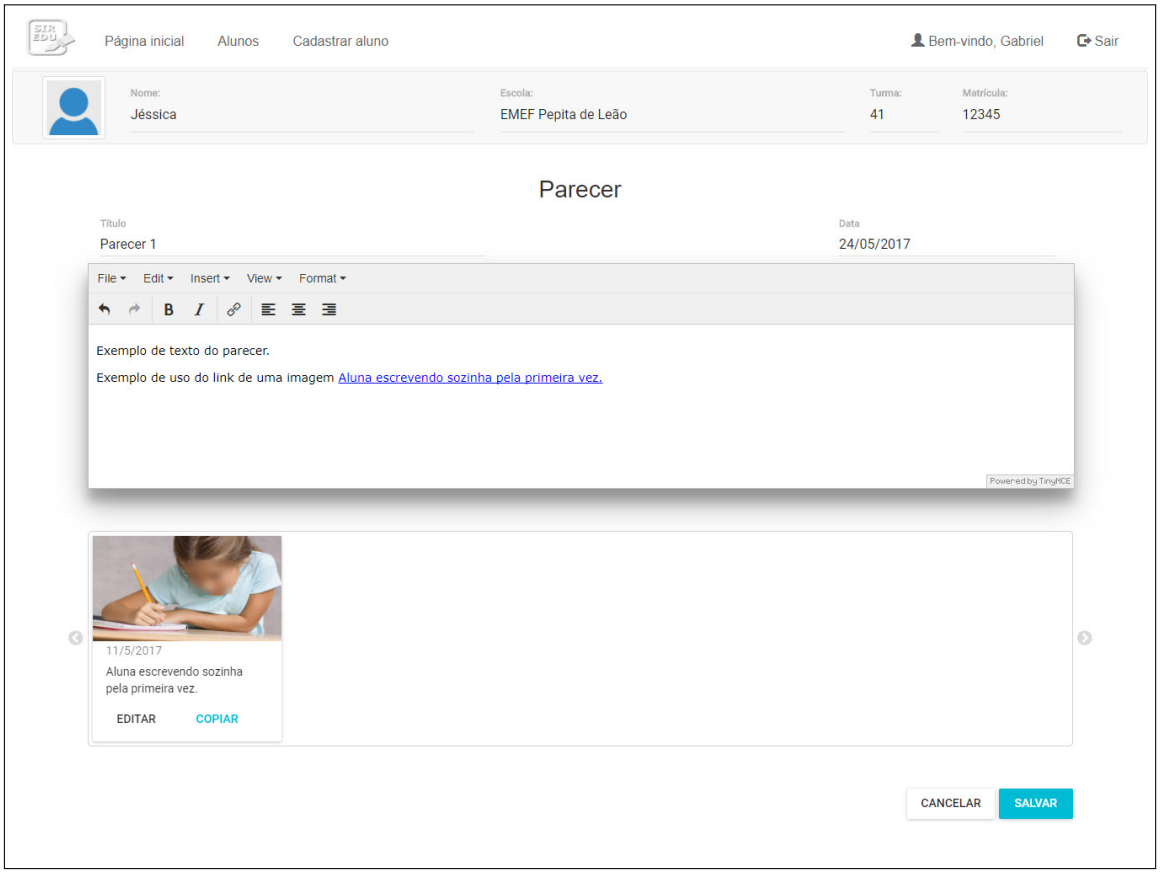
6.1.7 Tela de registro de pareceres

Ao clicar no item ”Parecer” no menu de registro do aluno, o professor logado é redirecionado para a página de criação de pareceres. No início desta página presenta-se dois campos para inserção de um título e data do parecer. Logo abaixo tem-se uma caixa

de edição de texto que, se necessário, expande conforme o usuário digita. Abaixo da área de texto está presente a linha do tempo de um aluno contendo as imagens ou vídeos previamente cadastrados. Cada arquivo da linha do tempo possui dois botões, um para edição e outro para copiar o link do arquivo para dentro do corpo do texto. Dessa forma, o professor pode analisar as atividades realizadas com o aluno e utilizá-las em tempo de criação do documento. Por fim, no rodapé da página, estão presentes os botões para voltar ao menu de registro ou salvar o parecer.

A Figura 6.11 ilustra um exemplo de preenchimento de um parecer. Pode-se observar que o link para a foto da aluna é utilizado no corpo do texto, destacado na cor azul. É importante ressaltar que o link está no formato do comentário presente na foto, essa funcionalidade serve para que o contexto da imagem possa ser usado na criação do texto do parecer. Ao clicar nesse link, uma nova janela do navegador é aberta contendo a imagem da aluna. Esta funcionalidade é necessária para que, após a geração de um documento PDF do parecer, seja possível visualizar a imagem fora do escopo da aplicação.

Figura 6.11: Exemplo de criação de parecer



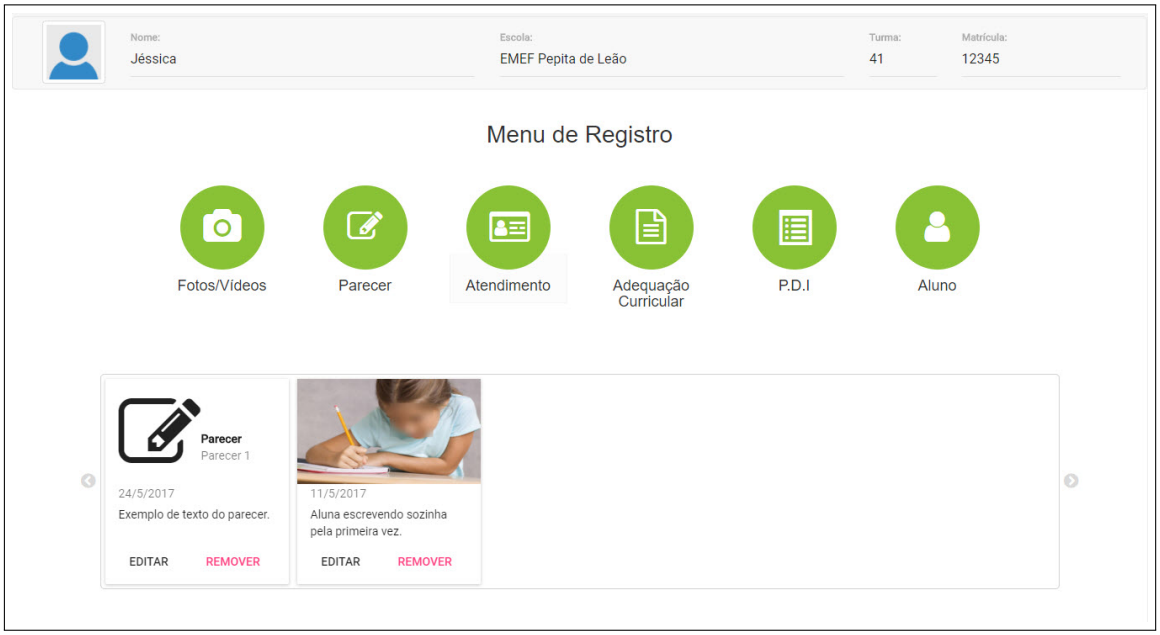
Fonte: Elaborada pelo autor

6.1.8 Linha do tempo

Além de estar presente na página de registro de pareceres, a linha do tempo de um aluno também pode ser gerenciada na página de menu de registro, como pode-se observar na Figura 6.12. Após registrar uma foto/vídeo ou um parecer, esses registros são inseridos na linha do tempo do aluno. Ela organiza na ordem cronológica o registro mais recente ao mais antigo. Nesse exemplo tem-se o parecer criado na Subseção 6.17 e o arquivo inserido na Subseção 6.16. Cada item da linha do tempo possui dois botões de ações, um para edição do registro e outro para remoção da base de dados. Um item não pode ser removido na página de registro de pareceres, apenas no menu do aluno.

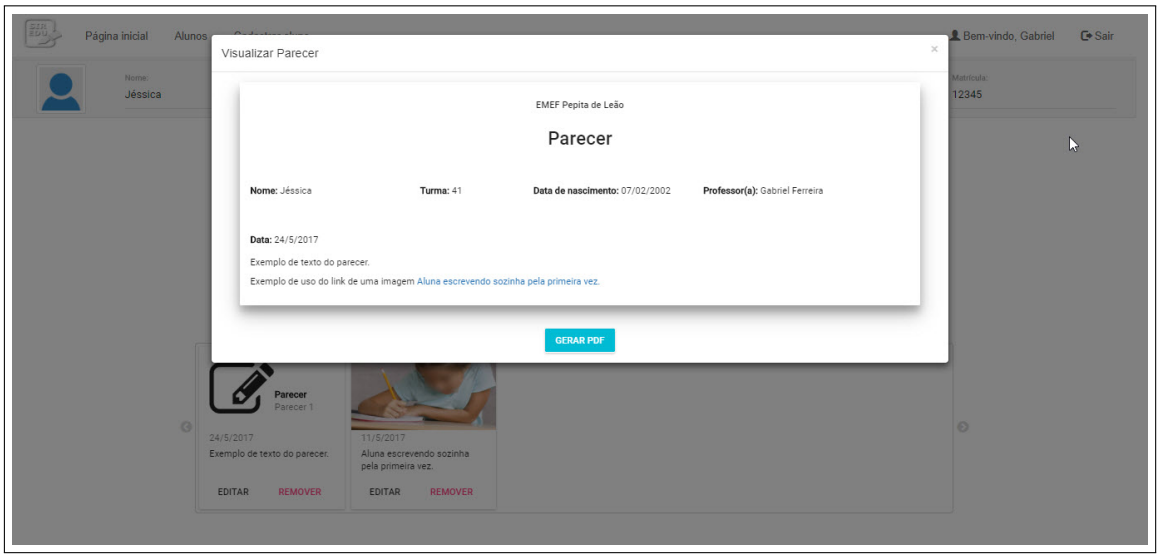
Ao clicar sobre um desses itens, uma função de visualização é ativada exigindo para usuário uma janela com informações relativas ao registro clicado. Na Figura 6.13 tem-se a janela de visualização do parecer com a opção de geração de um arquivo PDF. A funcionalidade de geração de PDFs não foi implementada nesta versão do protótipo. Na Figura 6.14 exibe-se a visualização da imagem cadastrada num tamanho maior.

Figura 6.12: Linha do tempo de um aluno



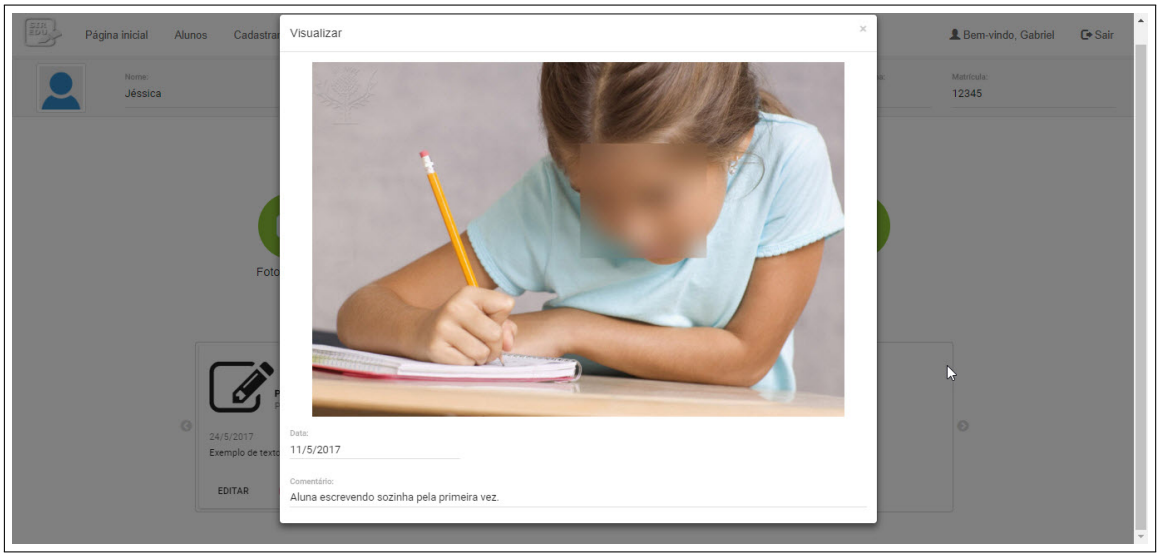
Fonte: Elaborada pelo autor

Figura 6.13: Visualização de um parecer



Fonte: Elaborada pelo autor

Figura 6.14: Visualização de um arquivo



Fonte: Elaborada pelo autor

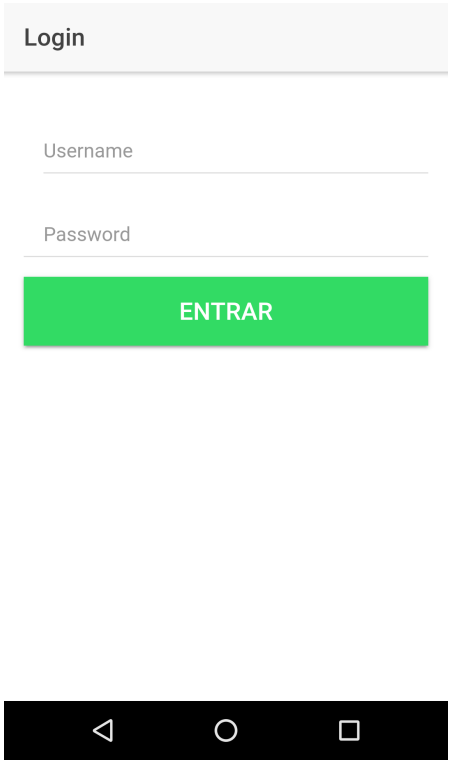
6.2 Aplicativo mobile

Nesta seção são apresentadas as telas correspondentes ao aplicativo *mobile* (móvel). São detalhadas as diferenças de funcionalidades em cada um dos protótipos, web e *mobile*, explicando cada funcionalidade com imagens capturadas da tela do dispositivo.

6.2.1 Tela de login

Para ter acesso ao aplicativo *mobile* é necessário ter uma conta com usuário e senha cadastrados na base de dados. Esse cadastro só é possível de ser efetuado através da tela de cadastro do sistema web. Após o usuário se cadastrar com sucesso, basta inserir seu e-mail e senha na tela de login e pressionar o botão ”Entrar” para realizar a autenticação. Pode-se observar que a API é a mesma compartilhada entre o sistema web e o aplicativo, pois o usuário consegue acessar os dois protótipos com a mesma conta.

Figura 6.15: Tela de login



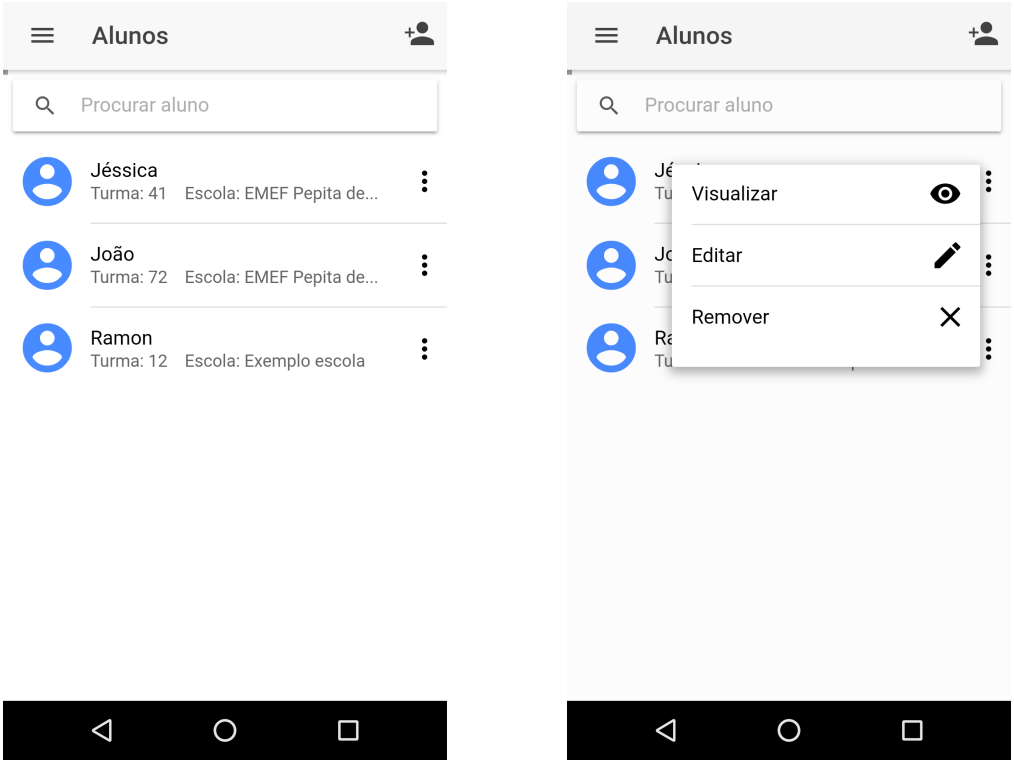
Fonte: Elaborada pelo autor

6.2.2 Lista de alunos

Ao logar-se na aplicação, o professor é redirecionado para a página inicial de listagem de seus alunos. No cabeçalho desta página tem-se dois ícones, o mais à esquerda ativa o menu lateral do aplicativo e o mais à direita redireciona para a página de cadastro de novos alunos. Logo abaixo exibe-se um campo de busca para filtrar um aluno da lista. Em seguida tem-se a listagem dos alunos.

Cada item da lista representa o mesmo aluno existente no sistema web, porém para ativar as ações possíveis para cada aluno, é necessário clicar no botão com o símbolo de elipses verticais. Ao clicá-lo, um menu na forma de *pop-over* é mostrado ao usuário, contendo as ações de visualização, edição e remoção de um aluno.

Figura 6.16: Tela da lista de alunos



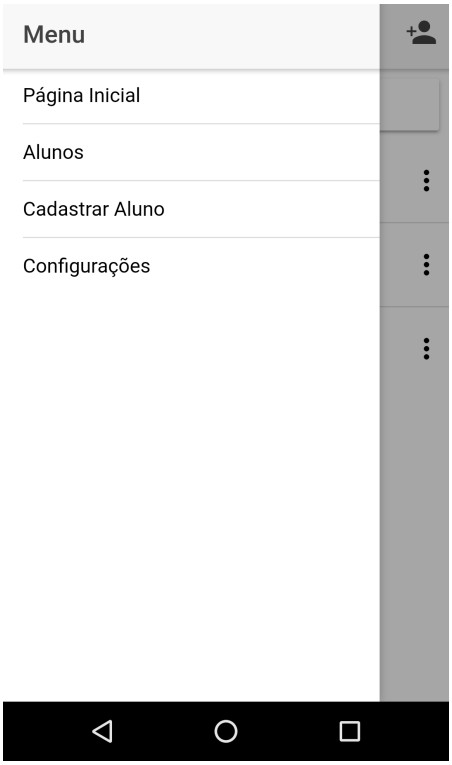
Fonte: Elaborada pelo autor

6.2.3 Menu lateral

O menu do aplicativo pode ser acessado por qualquer página que contenha o ícone de menu no seu cabeçalho. Ao ativá-lo, uma camada se sobrepõe sobre a tela atual, exibindo ações de navegação para a lista de alunos, cadastro de novos alunos ou configura-

ções gerais do aplicativo.

Figura 6.17: Menu de opções



Fonte: Elaborada pelo autor

6.2.4 Cadastro de alunos

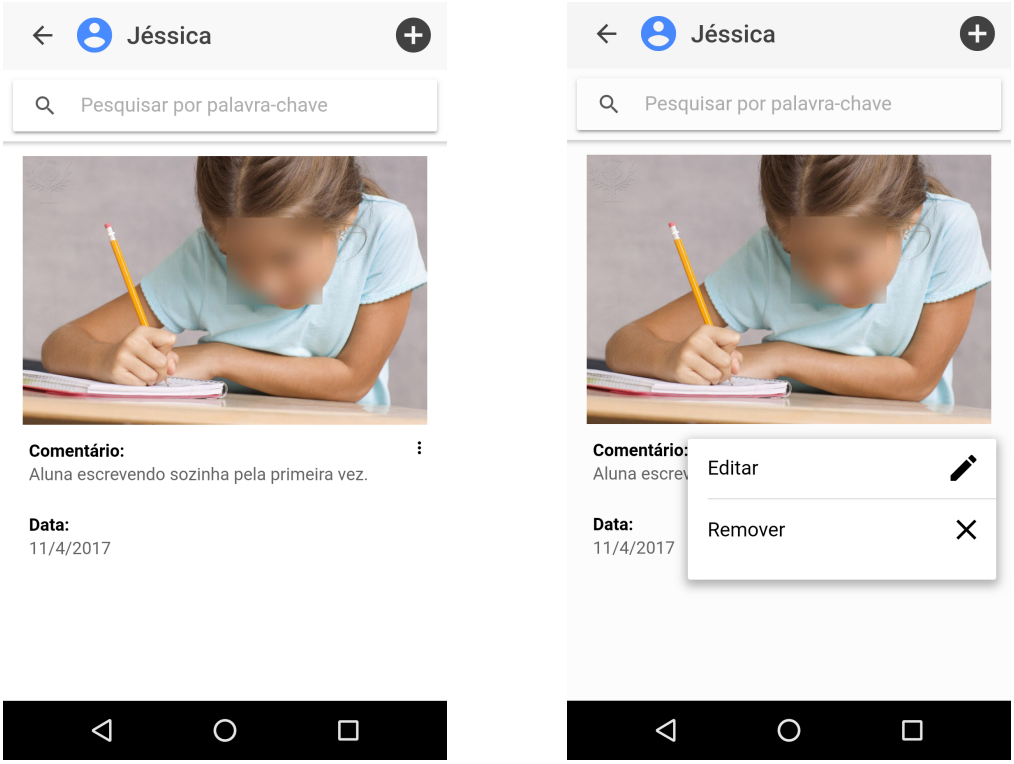
A tela de cadastro de alunos é similar à do sistema web, as mesmas informações são necessárias para preenchimento dos campos do cadastro de um aluno, a diferença é que o cadastro se torna mais extenso por não ser feito em etapas. Portanto, tem-se uma barra de rolagem na página para navegar entre esses campos. Outras funcionalidades como a de ”arrastar e soltar” um arquivo, não estão disponíveis por causa das características de um aplicativo *mobile* serem diferentes de um navegador web convencional.

6.2.5 Lista de arquivos

Ao clicar num aluno da lista de alunos, o usuário é redirecionado para a página de listagem de arquivos cadastrados para este aluno. No cabeçalho da página, à esquerda, tem-se uma seta indicando a ação de voltar para a lista de alunos, no centro é exibida a foto do aluno e seu nome, na ponta direita tem-se um ícone com o símbolo de adição, que ao ser clicado, redireciona o usuário para a página de adição de arquivos.

Abaixo do cabeçalho há um campo de busca para filtrar arquivos e em seguida exibe-se os itens da lista. No exemplo da Figura 6.18, tem-se a foto cadastrada anteriormente no sistema web, nela observa-se a imagem cadastrada e as informações relativas ao comentário e data sobre o acontecimento. Caso o tipo de arquivo seja um vídeo, o campo onde está a imagem se torna um *player* de vídeos HTML embutido na tela, exibindo os controles de resumir, pausar, controle de volume e modo *fullscreen* da mídia. Seguindo o padrão de elipses verticais do aplicativo, para exibir ações sobre um item, também tem-se o *pop-over* de ações para um item da lista de arquivos, nele tem-se as opções de edição e remoção do arquivo.

Figura 6.18: Tela da lista de arquivos de um aluno



Fonte: Elaborada pelo autor

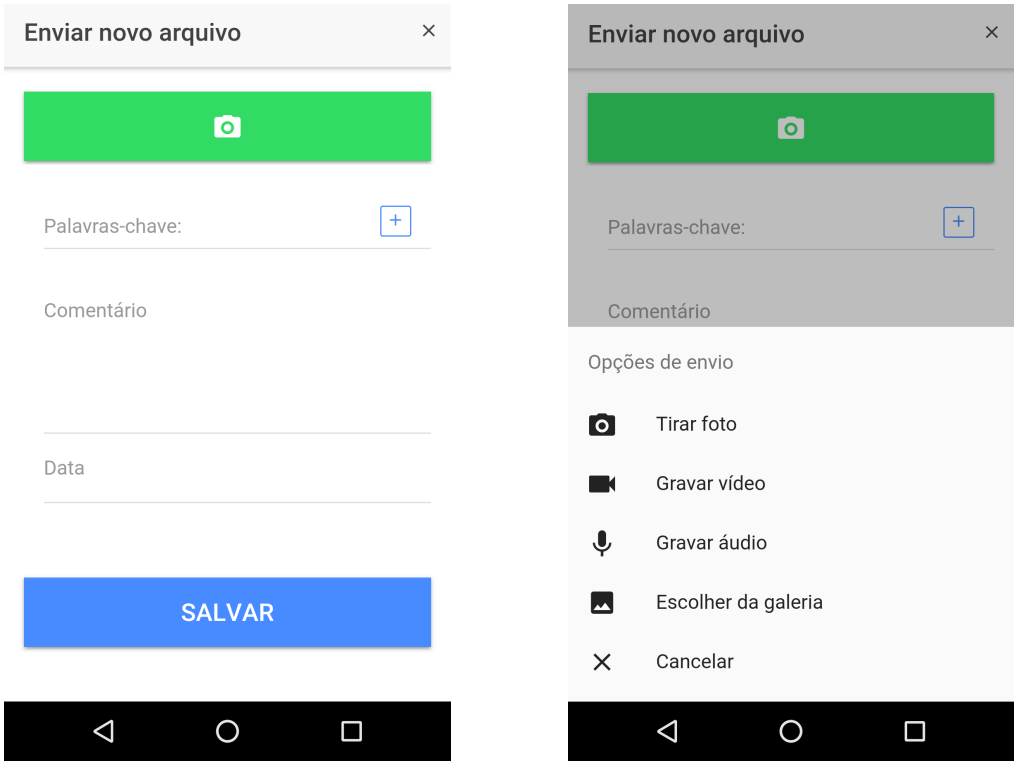
6.2.6 Tela de criação e adição de mídias

O cadastro de arquivos de mídia é a atividade principal do aplicativo mobile, pois ele estende a funcionalidade de enviar arquivos, possibilitando ao usuário capturar uma foto/vídeo/áudio e enviá-la diretamente de seu dispositivo para a base de dados. Na Figura 6.19 exibe-se a ela de envio de arquivos. Nela pode-se observar que há os campos de data e comentários exatamente como no sistema web. Além desses campos, há uma nova funcionalidade que permite ao usuário inserir palavras-chave que identificam essa foto, e elas são úteis para poder agrupar arquivos que possuem os mesmos critérios de busca. Para envio de um arquivo é necessário pressionar o botão verde com o símbolo de uma câmera. Ao clicar no botão, um menu de opções surgirá a partir da parte inferior da tela, neste menu são exibidas as possíveis formas de envio de um arquivo, que são as seguintes:

- **Tirar foto:** esta opção irá ativar o aplicativo padrão de câmera do dispositivo, após o usuário capturar uma imagem, ela será inserida na tela de envio de arquivos.
- **Gravar vídeo:** esta opção também irá ativar o aplicativo padrão de câmera do dispositivo, porém a câmera será ativada no modo captura de vídeos. Após a captura, o vídeo é inserido na tela de envio de arquivos.
- **Gravar áudio:** esta opção não está totalmente implementada, pois há problemas de compatibilidade com os sistemas operacionais diferentes dos dispositivos.
- **Escolher da galeria:** esta opção irá abrir o aplicativo de galeria do sistema operacional, após o usuário escolher a imagem, ela será inserida na tela de cadastro de arquivos.
- **Cancelar:** esta opção fecha o menu de opções retornando para a tela de cadastro.

Após o cadastro com sucesso de um arquivo, este estará disponível na lista de arquivos de um aluno. Como a base de dados é compartilhada, este mesmo arquivo também estará disponível na linha do tempo do aluno no sistema web.

Figura 6.19: Tela de registro de arquivos



Fonte: Elaborada pelo autor

6.3 Avaliação dos Sistemas

Após a implantação em produção do sistema web e do aplicativo móvel, os professores especializados que idealizaram o sistema testavam semanalmente as funcionalidades sendo inseridas. As conversas frequentes possibilitaram uma elaboração mais informal dos requisitos de cada *user story*. Foi através dessas conversas que foi identificado a necessidade de desenvolver algumas melhorias para deixar a navegação mais intuitiva para os usuários. Entre essas melhorias estão: complementar o sistema com mensagens informativas de erro ou sucesso quando o usuário realiza ações na tela, atualmente apenas algumas mensagens são disponibilizadas. Além dessas melhorias, foi pedido um reajuste da disposição dos ícones em algumas telas do aplicativo móvel e também a substituição de botões compostos por símbolos informativos para botões contendo um texto que indica a ação a ser executada ao clicá-lo.

No período final deste trabalho, um questionário foi elaborado para verificar a satisfação dos usuários e também para coletar sugestões de melhorias no sistema como um todo. O questionário é composto por um pré-experimento onde são solicitadas informações como idade, sexo, grau de experiência com web e aplicativos móveis, profissão e se o usuário trabalha no campo de educação especial. Após o preenchimento destas informações, é feito um experimento com os dois sistemas, web e móvel, para realizar as principais tarefas de acordo com as histórias de usuário definidas. Ao final de cada tarefa é solicitada a opinião do usuário com relação ao grau de dificuldade para sua execução. Por fim, foram disponibilizados campos para sugestões de melhorias e uma avaliação final para identificar se os protótipos ajudariam, ou não, no fluxo do trabalho de um professor especializado. No entanto, não foram coletadas respostas suficientes para serem levadas em consideração para análise. A sequência deste trabalho consistirá no reenvio desse questionário para um número maior de professores especializados, para então poder ser feita uma análise das respostas e levantar as possíveis melhorias a serem discutidas e incorporadas no sistema.

7 CONCLUSÃO

Este trabalho apresentou o projeto de arquitetura e desenvolvimento de dois protótipos com propósitos distintos, mas que, em conjunto, suprem as necessidades dos professores especializados, que são o público-alvo dos sistemas. O objetivo dos protótipos foi de propor uma maneira de os professores acompanharem de forma incremental o desempenho dos seus alunos com necessidades especiais, para que com base em suas análises pedagógicas, um plano de ação individual pudesse ser traçado para cada um dos alunos. Armazenar os dados digitalmente foi a motivação principal para que este trabalho pudesse ser realizado. As funcionalidades que resultaram em *user stories* foram elaboradas com base numa série de discussões com os professores orientadores. Alguns dos requisitos foram deixados de fora diante do curto espaço de tempo e complexidade para implementá-los, outros sofreram adaptações para poderem ser viáveis dentro do escopo definido. Um dos assuntos recorrentemente discutidos era com relação a segurança da informação contida no sistema, assim como proteção do uso de imagem dos alunos. A API do sistema foi projetada para somente dar acesso aos seus recursos se o usuário está autenticado e possuir as permissões necessárias. No entanto, há muito espaço para melhorias e discussões abertas para serem retomadas. De forma geral, os resultados alcançados neste trabalho se mostraram plenamente satisfatórios, pois foram implementadas todas as *user stories* definidas. Com relação aos aspectos técnicos, a decisão de utilizar apenas uma linguagem de programação durante todo projeto foi crucial para o seu andamento. Isso permitiu a reutilização de conhecimentos, dando velocidade e mais tempo para focar nas funcionalidades desejadas. Apenas com a utilização do *JavaScript*, e seus *frameworks*, foi possível implementar um sistema web, aplicativo mobile e *API REST*, o que comprova o poder desta linguagem. A API do sistema, embora não esteja completa por ser apenas um protótipo, deixa espaço para muitas possibilidades de trabalhos futuros. Isso ocorre pois ela pode ser facilmente integrada e utilizada em outras aplicações. Neste trabalho, por exemplo, tem-se duas aplicações diferentes (sistema web e aplicativo *mobile*) acessando a mesma base de dados através da *API REST*. Como mencionado anteriormente, essa abordagem de expor a API exigiria uma segurança maior das informações. Portanto fica a cargo de trabalhos futuros uma análise mais profunda e implementação de uma camada de segurança robusta capaz de suprir todas as necessidades legais do sistema. Uma outra sugestão de trabalhos futuros seria de dar continuidade ao desenvolvimento dos protótipos. Devido ao tempo limitado, ficaram de fora funcionalidades importantes como: geração de

documentos *PDF* e *Word*, registro de atendimentos de aluno, planos de desenvolvimento individuais, programação, análises curriculares e a partir dessas informações salvas, possibilitar a geração de gráficos com os dados disponíveis. Em vista disso, seria interessante ter essas funcionalidades disponíveis para tornar os sistemas mais completos.

REFERÊNCIAS

CHROME. **Chrome V8 - Introduction**. 2017. <https://developers.google.com/v8/>. [Online: Acessado em 23 de Junho de 2017].

CODESCHOOL. **Single-page Applications**. 2017. <https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications>. [Online: Acessado em 18 de Junho de 2017].

COHN, M. **User stories applied: For agile software development**. [S.l.]: Addison-Wesley Professional, 2004.

CORDOVA, A. **Overview**. 2017. <https://cordova.apache.org/docs/en/7.x/guide/overview/>. [Online: Acessado em 21 de Junho de 2017].

EXPRESS. **Roteamento**. 2017. <http://expressjs.com/pt-br/guide/routing.html>. [Online: Acessado em 25 de Junho de 2017].

FIELDING, R. **Representational State Transfer (REST)**. 2000. <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. [Online: Acessado em 25 de Junho de 2017].

FLUX. **In Depth Overview**. 2015. <https://facebook.github.io/flux/docs/in-depth-overview.html#content>. [Online: Acessado em 18 de Junho de 2017].

IONIC. **Core Concepts**. 2017. <https://ionicframework.com/docs/intro/concepts/>. [Online: Acessado em 21 de Junho de 2017].

JSON.ORG. **Introducing JSON**. 2017. <http://www.json.org/>. [Online: Acessado em 16 de Junho de 2017].

JWT.IO. **Introduction**. 2017. <https://jwt.io/introduction/>. [Online: Acessado em 27 de Junho de 2017].

MONGODB. **GridFS**. 2017. <https://docs.mongodb.com/manual/core/gridfs/>. [Online: Acessado em 28 de Junho de 2017].

MONGODB. **MongoDB Architecture**. 2017. <https://www.mongodb.com/mongodb-architecture>. [Online: Acessado em 27 de Junho de 2017].

MONGODB. **What is MongoDB?** 2017. <hhttps://www.mongodb.com/what-is-mongodb>. [Online: Acessado em 27 de Junho de 2017].

MONGOOSE. **Mongoose - elegant mongodb object modeling for node.js**. 2017. <http://mongoosejs.com/>. [Online: Acessado em 30 de Junho de 2017].

MOZZILA. **AJAX - Getting Started**. 2017. <https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started>. [Online: Acessado em 13 de Junho de 2017].

MOZZILA. **HTML**. 2017. <https://developer.mozilla.org/en-US/docs/Web/HTML>. [Online: Acessado em 10 de Junho de 2017].

MOZZILA. **JavaScript**. 2017. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. [Online: Acessado em 10 de Junho de 2017].

NODE.JS. **About Node.js**. 2017. <https://nodejs.org/en/about/>. [Online: Acessado em 23 de Junho de 2017].

NPM. **What is npm?** 2017. <https://www.npmjs.com>. [Online: Acessado em 24 de Junho de 2017].

REACT. **A JavaScript library for building user interfaces**. 2017. <https://facebook.github.io/react/>. [Online: Acessado em 15 de Junho de 2017].

REDUX. **Basics Guide**. 2017. <http://redux.js.org/docs/basics/>. [Online: Acessado em 18 de Junho de 2017].

W3C. **A Short History of JavaScript**. 2012. <https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScrip>. [Online: Acessado em 5 de Junho de 2017].

WEBPACK. **Concepts**. 2017. <https://webpack.js.org/concepts/>. [Online: Acessado em 12 de Junho de 2017].