UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Luan Thome Correa

Desenvolvimento de uma versão Mobile do Portal do Aluno da UFES

Vitória, ES

2022

Luan Thome Correa

Desenvolvimento de uma versão Mobile do Portal do Aluno da UFES

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Ciência da Computação

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2022

Luan Thome Correa
Desenvolvimento de uma versão Mobile do Portal do Aluno da UFES/ Luan Thome Correa. – Vitória, ES, 2022-
70 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Colegiado do Curso de Ciência da Computação, 2022.

1. Palavra-chave1. 2. Palavra-chave2. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Desenvolvimento de uma versão Mobile do Portal do Aluno da UFES

CDU 02:141:005.7

Luan Thome Correa

Desenvolvimento de uma versão Mobile do Portal do Aluno da UFES

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, (dia) de (mês) de (ano):

Prof. Dr. Vítor E. Silva Souza
Orientador

Camila Zacche de Aguiar
Universidade Federal do Espírito Santo

Symone de Deus Miranda Gonçalves
Universidade Federal do Espírito Santo

Vitória, ES
2022

Dedico este trabalho à todos os familiares e amigos que de alguma forma estiveram ao meu lado em todos esses anos de graduação.

Agradecimentos

Primeiramente gostaria de agradecer a todos os familiares que me apoiaram desde o início em minha decisão de seguir com esse curso, em especial aos meus pais que, além de me formarem como pessoa da melhor maneira possível, também fizeram todo o possível para que eu tivesse condições de ingressar e me manter no curso.

Também gostaria de agradecer a todos os colegas e amigos que tive a oportunidade de conhecer, trocar informações, e pelos vários suportes recebidos durante todo o curso que foram fundamentais para a conclusão da graduação. Levo definitivamente comigo diversas lembranças boas de vários momentos vivenciados na Universidade.

Agradeço também à todos os professores que de alguma forma contribuíram para o ganho de conhecimentos, muitos necessários para a vida e para o ramo profissional. Em especial agradeço ao meu orientador Vítor, que além de uma pessoa incrível, me orientou com muita leveza em todo o processo de conclusão do curso, como também em outros trabalhos de extensão.

Por fim, agradeço à Deus, por todo o conhecimento e sabedoria que me permitiram construir toda uma caminhada que me permitisse concluir mais um ciclo na vida.

"Tente mover o mundo – o primeiro passo será mover a si mesmo.
(Platão)

Resumo

O portal do aluno de uma Universidade é uma ferramenta essencial no acompanhamento do processo acadêmico, pois através dele é possível realizar diversas atividades, como matrículas, trancamentos, etc. Além disso, ele também serve para fornecer e visualizar documentos essenciais em diversos processos dentro e fora do escopo da Universidade.

O portal do aluno Web construído para a UFES, em termos de funcionalidade, apresenta o comportamento esperado. É possível navegar nele e fazer o uso de suas funcionalidades sem maiores problemas. Entretanto, em termos de experiência do usuário, este portal apresenta alguns problemas principalmente quando acessado por meio de um dispositivo móvel. Os componentes da plataforma não se adequam de maneira responsiva à tela do *smartphone*, o que prejudica a experiência de uso. Além disso, este portal também carece de algumas funcionalidades que poderiam servir para auxiliar ainda mais os alunos da UFES.

Neste trabalho foi, portanto, desenvolvido um aplicativo para dispositivos móveis que representa uma versão do portal do aluno da UFES já existente, mas com algumas melhorias em sua experiência de uso e adição de novas funcionalidades. Através de processos de Engenharia de Software, foi realizado um levantamento de requisitos e modelagem de diagramas, baseados tanto em funcionalidades já existentes no portal do aluno da UFES, como para novas funcionalidades. Por fim foi implementado o aplicativo utilizando a abordagem *Cross-Platform* e como tecnologia principal o React Native, que permite construir aplicações para as plataformas iOS e Android simultaneamente.

Palavras-chaves: Desenvolvimento Cross-platform, React Native, iOS, Android.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UML	Unified Modeling Language
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
ECMA	European Computer Manufacturers Association
TSC	TypeScript Compiler
MVC	Model-View-Controller
DOM	Document Object Model
JSX	JavaScript XML
XML	Extensible Markup Language
API	Application Programming Interface
JSON	JavaScript Object Notation
NoSQL	Not Only Structured Query Language
IDE	Integrated Development Environment
SDK	Software Development Kit
RU	Restaurante Universitário
PDF	Portable Document Format

Sumário

1 Introdução

O uso de aplicativos de dispositivos móveis transformou a forma como nos comunicamos, entretemos e organizamos. É bastante comum encontrarmos, nos dias atuais, a substituição de padrões utilizados há muito tempo na sociedade, por novos padrões, mas agora aplicados ao meio digital. Quando alguém diz, por exemplo, que irá "consultar sua agenda" para fins de disponibilidade, pode existir a dúvida sobre qual meio esta pessoa utilizará para realizar essa consulta, se em uma caderneta ou em um *smartphone*, visto que ambas as situações se tornaram comuns.

Os aplicativos móveis são construídos com uma gama variada de objetivos. Muitos destes objetivam fornecer aos usuários, maneiras de organizar e monitorar sua vida, como observado em (EL-GAYAR et al., 2013), onde foi desenvolvida uma aplicação para monitoramento de diabetes. Mas também existem aplicativos móveis que se tratam de adaptações de uma versão *Web* existente. Muitas vezes essas adaptações são necessárias pois a forma pela qual se utiliza um *smartphone* e um computador são diferentes e requerem diferentes tratamentos.

A UFES (Universidade Federal do Espírito Santo) dispõe de um portal digital oferecido aos alunos, onde é possível realizar diversas atividades essenciais à vida acadêmica, bem como consultar variados históricos individuais. É possível utilizar o portal sem maiores problemas quando acessado por um computador, entretanto, se utilizado através de um dispositivo móvel, será possível notar certa falta de responsividade na aplicação, ou seja, o portal não se adéqua corretamente à tela do dispositivo utilizado. Essa falta de responsividade não permite que o usuário tenha uma experiência fluida ao acessar o portal, e necessite realizar adaptações manuais para conseguir visualizar melhor os elementos do portal e acessar as seções desejadas. A Figura 1 traz um exemplo dessa falta de responsividade, na qual observa-se que os elementos acessados são muito pequenos comparados ao tamanho da tela do *smartphone*, e faz-se necessário ampliar a mesma para se obter uma melhor experiência.

O portal do aluno da UFES, por vezes, é alvo de discussões a respeito de melhorias e novidades que este poderia receber. Podem ser citados como melhorias/novidades:

- Uma possível integração com o sistema de saldo do RU (Restaurante Universitário) da UFES, visto que a única maneira de consultar esse saldo seria se locomover até uma máquina eletrônica situada no restaurante;
- Um *feed* de notícias integrado, o qual permitiria os colegiados enviarem mensagens e se comunicarem com os estudantes, criando um ambiente de comunicação mais

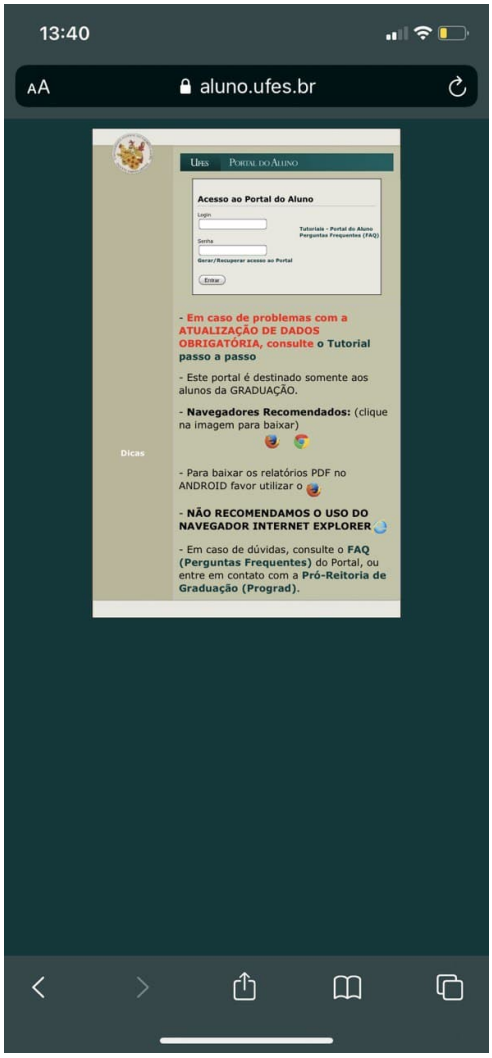

Figura 1 – Portal do Aluno acessado por um *smartphone*.

dinâmico e atraente aos jovens;

- A possibilidade de serem enviadas notificações aos estudantes das atividades que possuem um determinado prazo, pois apesar de ser essencial os estudantes terem ciência desse prazo, essa funcionalidade poderia ajudar a evitar complicações e esquecimentos.

O problema da responsividade do portal do aluno da UFES, citado em parágrafo anterior, atrelado à observação de melhorias que este poderia receber, motivaram a criação deste trabalho. A ideia é desenvolver um sistema que garanta uma boa experiência aos usuários que o acessarem, além de permitir novos meios de organizar sua vida acadêmica.

1.1 Objetivos

O objetivo geral deste projeto consiste em construir um aplicativo Mobile, que é baseado no portal do aluno da UFES em sua versão Web. Neste aplicativo procura-se criar

uma experiência de uso em *smartphones*, que seja melhorada em relação à experiência vivenciada na versão Web já existente do portal do aluno. Além disso, também fazem parte do objetivo deste trabalho a incorporação de novas funcionalidades que foram pensadas para facilitar a vida acadêmica dos alunos da UFES.

São objetivos específicos deste projeto:

- Levantar e documentar os requisitos necessários tanto para a base da aplicação, quanto para as novas funcionalidades;
- Definir a arquitetura a ser utilizada, produzindo posteriormente o documento de projeto do sistema;
- Implementar o sistema baseando-se no documento de projeto definido, fazendo o uso de *frameworks* e ferramentas já existentes, que auxiliarão no desenvolvimento.

1.2 Método de Desenvolvimento do Trabalho

O método de desenvolvimento deste trabalho será composto pelas seguintes atividades:

- Revisão bibliográfica: estudo de assuntos relevantes e necessários para apoiar o desenvolvimento do sistema, tais como Engenharia de Software, linguagem de programação JavaScript, *frameworks* JavaScript para o desenvolvimento de aplicações *Web* e *Mobile*, entre outros.
- Levantamento e especificação de requisitos: fase onde serão primeiramente levantados todos os requisitos observados para o sistema e que posteriormente serão documentados na especificação, fazendo o uso de diagramas que ilustram de forma clara suas relações.
- Documento de projeto: fase onde será elaborado uma documentação contendo tecnologias e *frameworks* utilizados no desenvolvimento do sistema, bem como sua arquitetura detalhada através de diagramas.
- Implementação: fase onde será desenvolvido todo o sistema, tomando como base toda a documentação realizada nas atividades anteriores. Será desenvolvido um protótipo e alguns testes serão feitos no mesmo.
- Monografia: como última etapa de desenvolvimento desse trabalho, será produzida uma monografia contendo toda a descrição do mesmo. Será feito o uso da ferramenta *TeXstudio*, e o modelo de documento *abnTeX2*.

1.3 Organização do Texto

Esta monografia possui cinco capítulos e está dividida da seguinte maneira:

- **Capítulo** 1 - Introdução: apresenta uma contextualização geral deste projeto, contendo objetivos e métodos utilizados para o desenvolvimento deste trabalho;
- **Capítulo** 2 - Fundamentação Teórica e Tecnologias utilizadas: apresenta uma revisão da literatura acerca dos temas relevantes ao contexto deste trabalho e introduz as principais ferramentas utilizadas no desenvolvimento deste projeto;
- **Capítulo** 3 - Especificação de Requisitos: descreve a especificação de requisitos do sistema, contendo o escopo, tabelas de requisitos e diagramas de casos de uso e de classes;
- **Capítulo** 4 - Projeto Arquitetural e Implementação: apresenta a arquitetura do sistema, assim como detalhes dos principais tópicos de sua implementação, ilustradas com uma sequência de capturas de tela;
- **Capítulo** 5 - Considerações Finais: apresenta as conclusões do trabalho, dificuldades e possíveis melhorias para trabalhos futuros.

2 Fundamentação Teórica e Tecnologias Utilizadas

Essa seção é destinada à apresentação de conceitos, tecnologias e ferramentas utilizadas no desenvolvimento do sistema proposto, e será dividida nas seguintes subseções:

- Na Seção 2.1, encontram-se conceitos relacionados à Engenharia de Software;
- Na Seção 2.2, serão apresentados alguns conceitos de Desenvolvimento Web, e *frameworks* utilizados;
- Na Seção 2.3, será apresentado o conceito de Desenvolvimento Multi-plataforma, e os *frameworks* escolhidos para tal.

2.1 Engenharia de Software

Com o avanço da tecnologia, que permitiu o desenvolvimento de softwares cada vez mais complexos, surgiu uma área de estudo preocupada em organizar e conduzir todo o processo de criação de um software. Essa área é a Engenharia de Software, que define etapas e padrões a serem seguidos para assegurar uma boa qualidade e confiabilidade das aplicações. Nas sub-seções a seguir, serão descritas suas principais práticas.

2.1.1 Especificação de Requisitos

A primeira etapa no desenvolvimento de um software, consiste na definição dos requisitos que esse sistema busca atender. Primeiramente deve-se realizar o **levantamento de todos os requisitos** necessários. Ainda nessa etapa, se faz necessário produzir uma descrição que seja suficiente para definir o escopo desse sistema. Tendo posse desse material, busca-se então refinar e detalhar de maneira mais precisa os requisitos, atribuindo restrições aos mesmos, bem como definindo suas funções e medidas de desempenho (FALBO, 2014). Essa primeira etapa define uma base importante para o software em questão, que busca além de outros objetivos, minimizar erros e inconsistências em seu futuro desenvolvimento. Ainda assim, é comum que essa etapa seja revisitada ao longo do seu desenvolvimento, permitindo que hajam condições para redefinir ou adicionar requisitos desejados.

Os requisitos podem assumir duas classificações distintas: funcionais e não funcionais. Os **requisitos funcionais** definem o que um determinado sistema deve fazer, mas não definem como este deve fazer. Existe uma variação na forma como esses requisitos são

expressados, a depender do tipo de software que será desenvolvido, os possíveis usuários que o utilizarão, e da abordagem utilizada pela equipe responsável por determinar os requisitos. Além disso, os requisitos funcionais buscam trazer características ao sistema como completude e consistência. No primeiro, busca-se cobrir todos os possíveis serviços requeridos pelo usuário, enquanto que o segundo têm como objetivo eliminar possíveis definições contraditórias que podem surgir ao longo do processo (SOMMERVILLE, 2011). São alguns exemplos de requisitos funcionais para um sistema de concessionárias:

- O usuário deve ser capaz de pesquisar todas as concessionárias presentes em sua cidade;
- O usuário deve ser capaz de checar quais veículos estão disponíveis por concessionária;
- O sistema deve gerar mensalmente, para cada concessionária, um relatório contendo todas as pesquisas realizadas pelos usuários.

Com sua alta importância no processo de especificação de requisitos, os **requisitos não funcionais** diferem-se dos funcionais justamente por não estarem preocupados com as funcionalidades específicas do sistema, oferecidas aos usuários. Estes requisitos, no entanto, se relacionam diretamente com propriedades mais críticas do sistema, como por exemplo confiabilidade, desempenho, ocupação de área, etc. Além disso, são comumente relacionados ao sistema de forma geral e não a componentes específicos, ou seja, há uma grande necessidade de se atender esses requisitos, visto que o caso contrário pode acarretar na inutilização do sistema (SOMMERVILLE, 2011). São alguns exemplos de requisitos não funcionais:

- O sistema deve ser compatível com os principais navegadores Web (portabilidade);
- O sistema não deve demorar mais do que 5 segundos para processar uma transação (desempenho);
- O sistema deve estar disponível para uso em todos os dias e a qualquer hora (disponibilidade).

Realizado o levantamento de requisitos, procede-se então para a próxima etapa, definida como **análise de requisitos**. Essa etapa visa tornar os requisitos encontrados consistentes e livre de ambiguidades, permitindo que estes sejam utilizados em futuras etapas do processo de desenvolvimento. Para representar esses conjuntos de requisitos, é utilizada a técnica de **modelagem conceitual**, que consiste em representar um sistema através do uso de modelos, considerando características que independem do ambiente computacional onde esse sistema será implementado (FALBO, 2014). Existem diversas técnicas de modelagem que podem ser utilizadas na modelagem conceitual, e abaixo encontram-se aquelas que serão utilizadas neste trabalho:

- **Modelagem de casos de uso**: representa todas as possíveis interações que estão descritas no documento de requisitos do sistema (SOMMERVILLE, 2011);
- **Modelagem de Entidades e Relacionamentos**: representa entidades do mundo real e seus possíveis relacionamentos, os quais são simulados internamente por um sistema (FALBO, 2014);
- **Modelagem de Estados**: descreve todos os possíveis estados que as entidades podem assumir, no decorrer de sua vida, bem como os eventos que provocam tal mudança de estado (FALBO, 2014).

Realizadas as etapas de levantamento e análise de requisitos descritas nesta mesma seção, têm-se então, informações suficientes que permitem a criação de um documento essencial nessa etapa do projeto, o **documento de requisitos**. Neste trabalho, esse documento será dividido em uma parte que agrupará os requisitos levantados e outra parte que irá conter a modelagem conceitual realizada com suas descrições.

2.1.2 Projeto de Software

A etapa de projeto ou design de software define como o sistema deverá ser implementado. A ideia é incorporar a tecnologia aos requisitos essenciais do usuário, que foram obtidos na etapa de especificação de requisitos mencionada anteriormente (FALBO, 2014). Essa pode ser considerada a última etapa da Engenharia de Software que envolve modelagem, permitindo assim, que finalmente se possa prosseguir para a fase de construção do sistema, assegurando qualidade para o mesmo (PRESSMAN, 2011).

Um sistema de software pode depender de vários outros sistemas para ter o seu funcionamento correto e consistente. É muito comum, por exemplo, a utilização de sistemas operacionais, banco de dados, *middlewares*, entre outras aplicações. Portanto, se torna essencial um bom conhecimento a respeito desses sistemas que serão utilizados, pois permite que os projetistas decidam a melhor forma de integrá-los ao sistema que se quer desenvolver (SOMMERVILLE, 2011).

Ainda de acordo com Sommerville (2011), diferentes tipos de sistemas podem demandar diferentes atividades no processo de projeto de software. As atividades que podem aparecer nesse processo podem ser definidas como:

- Projeto de Arquitetura do Software: tem como objetivo definir os componentes estruturais do software, bem como seus relacionamentos;
- Projeto de dados: tem como objetivo projetar a estrutura de armazenamento de dados que será necessária para implementação do sistema;

- Projeto de Interfaces: descreve as diferentes comunicações que o sistema realiza, sejam elas internas, externas ou com os usuários;
- Projeto Detalhado: visa refinar e detalhar a descrição dos componentes estruturais apresentados na arquitetura do software.

Para cada uma destas sub-etapas, são utilizados diferentes tipos de modelos que as representam, e que novamente serão reunidas em documentos. Neste trabalho, os documentos referentes à cada uma dessas atividades serão reunidos em um só, denominado **documento de projeto de sistema**.

2.2 Desenvolvimento Web

Apesar deste trabalho não se tratar de uma aplicação Web própriamente dita, essa seção descreve linguagens e ferramentas que, a principio, foram desenvolvidas para atender ao ambiente Web, mas que evoluiram para também apoiar o desenvolvimento *mobile*. Além disso, também será descrito um sistema necessário para realização de testes e simulações, que também faz uso de conceitos referentes à Web.

2.2.1 JavaScript

O JavaScript é a linguagem de programação da Web. A grande maioria dos sites modernos usam JavaScript em seu desenvolvimento, seja diretamente ou com o uso de *frameworks*. Além disso, grande parte dos navegadores modernos, utilizados por exemplo em *desktops*, jogos, *tablets*, incluem tradutores de JavaScript, o que permitiu a essa linguagem se tornar uma das mais onipresentes da história. Essa linguagem pode ser considerada parte de uma tríade que todo desenvolvedor Web deve aprender: HTML para especificação de conteúdo, CSS para especificação de apresentação e JavaScript para especificar o comportamento das páginas Web (FLANAGAN, 2011).

A linguagem JavaScript pode ser considerada como sendo de alto nível, dinâmica, não tipada e ideal para estilos de programação que envolvem orientação a objetos e programação funcional. Muitas linguagens utilizam notação de ponto para acessar propriedades e métodos em objetos e no JavaScript a sintaxe é a mesma. Entretanto, em grande parte das vezes não se faz necessário definir uma classe, importar pacotes ou incluir arquivos de cabeçalho. É possível começar a programar com determinados tipos de dados e, após isso, agrupá-los de diversas maneiras possíveis. Certamente é possível usar JavaScript de maneira procedural, mas seu verdadeiro potencial aparece quando se aproveita das vantagens de sua natureza orientada a objeto (ZAKAS, 2016).

De acordo com Flanagan (2011), o JavaScript foi criado pela empresa Netscape nos primeiros dias de surgimento da Web e, tecnicamente, “Java-script” é uma marca licenciada

da Sun MicroSystems (atualmente Oracle). A Netscape submeteu a linguagem para ser padronizada para o ECMA – Associação Europeia de Fabricantes de Computadores, porém, devido à complicações de marca registrada, o nome da linguagem ficou presa com o nome "ECMAScript". Pelas mesmas questões, a versão da Microsoft dessa linguagem é formalmente conhecida como "JScript". Na prática, todos chamam a linguagem de JavaScript.

No momento da escrita deste trabalho, a atual versão do ECMAScript encontra-se em sua 12ª edição. O JavaScript passou por diversas atualizações, e muitas delas permitiram o desenvolvimento de ferramentas que trouxeram diversas possibilidades diferentes para a criação de funcionalidades para a Web. Na Seção 2.2.3 será descrito um exemplo de biblioteca que surgiu em meio ao desenvolvimento do JavaScript.

2.2.2 TypeScript

Como mencionado na seção anterior, o JavaScript é uma linguagem não tipada, e em muitas aplicações a tipagem é algo que pode fazer a diferença em diversas esferas do desenvolvimento. Neste contexto, surge a linguagem TypeScript, que é definida por Shubel (2022) como um super conjunto da linguagem JavaScript, que adiciona uma sintaxe para tipos e contém outras funcionalidades adicionais.

Ainda Shubel (2022) define os três principais componentes do TypeScript:

- **Linguagem**: a sintaxe, palavras-chave e definições de tipos;
- **O Compilador TypeScript (TSC)**: converte as instruções escritas em TypeScript para o equivalente em JavaScript;
- **Serviço de Linguagem JavaScript**: uma camada adicional que auxilia na escrita do código em um editor de texto, implementando funcionalidades como completação de códigos, lembrete de assinatura, formatação de código, colorização, entre outros.

Coderslang (2021) descreve um outro ponto que pode ser resolvido pelo TypeScript. Um dos maiores problemas do JavaScript é que um problema só pode ser detectado em tempo de execução, e isso pode resultar em erros que afetam o usuário final. Isso pode afetar qualquer negócio negativamente pois prejudica a experiência do usuário. Entretanto, o TypeScript consegue resolver alguns desses problemas, pois este checa por problemas em tempo de compilação.

O código presente na Listagem 2.1 não acusará nenhum erro se usado o JavaScript, entretanto, se for usado TypeScript, o compilador irá acusar que a propriedade **nonExistentProperty** não existe no objeto **obj**.

Listagem 2.1 – Exemplo de erro tratado pelo TypeScript.

```
const obj = {
  foo: "baz",
};

console.log(obj.nonExistentProperty);
```

A Listagem 2.2 descreve um exemplo de uma tipagem feita com TypeScript, em comparação com o mesmo código na linguagem JavaScript. Dependendo do editor que estiver sendo utilizado, os tipos presentes podem ser sugeridos para facilitar o desenvolvimento.

Listagem 2.2 – Exemplo de erro tratado pelo TypeScript.

```
// Javascript

function setUserNameAndCPF(user, name, cpf) {
    user.name = name;
    user.cpf = cpf;
}

// Typescript

interface User {
  name: string;
  cpf: string;
}

function setUserNameAndCPF(user: User, name: string, cpf: string) {
    user.name = name;
    user.cpf = cpf;
}
```

2.2.3 React

Segundo Robbestad (2016), React não é considerado um *framework*. O React se descreve como o **V** que aparece no padrão MVC (*Model-View-Controller*), ou seja, a camada de visão. É uma biblioteca que pode ser combinada junto a outras, como o AngularJS, por exemplo, ou até mesmo com bibliotecas populares do JavaScript, como o Knockout. O React vêm estabelecendo certa consolidação no mercado, uma vez que têm sido utilizado tanto por empresas pequenas, quanto por grandes empresas como AirBnB, Uber, Netflix, entre outras.

O React é considerado uma biblioteca orientada a componentes, os quais podem ser agrupados e com a possibilidade de controlar seus respectivos estados. Esses componentes contêm tanto a logica para aplicação da camada de visão, quanto a camada de visão por si própria. A escolha do uso de componentes, aliada ao processo de renderização do React, o qual realiza uma renderização completa para aplicação, contribui para uma menor complexidade de interações na mesma, bem como elimina a necessidade de renderização sub-sequente (VIPUL et al., 2016).

Segundo Vipul et al. (2016), React se basea na ideia de que a manipulação do DOM[1] (*Document Object Model*), é uma operação custosa e deve ser minimizada. Além disso, a sua manipulação manual pode resultar em códigos repetitivos e desnecessários, que podem causar um grande impacto na performance da aplicação. A solução utilizada pelo React para contornar essa questão, é fornecer ao desenvolvedor um **DOM virtual** para realizar a renderização, ao invés do DOM original. Robbestad (2016) comenta que o DOM virtual mantém os conjuntos internos de elementos mais compactos e simplificados, e apenas são aplicadas mudanças para o DOM original, quando é alterado o estado de um conjunto de elementos. Isso permite alterar partes de elementos visíveis sem que outros elementos sejam afetados, provendo uma considerável eficiência nas atualizações do DOM.

Aplicações desenvolvidas com React podem ser escritas utilizando o formato **JSX**, que é uma mistura de JavaScript com a linguagem de marcação XML. Conforme o que é comentado por Holmes et al. (2015), esse formato facilita a construção de componentes pela sua característica declarativa, a qual descreve os elementos exatos que devem ser vistos em um determinado componente. Além disso, uma vez que a lógica pode estar contida dentro do próprio componente, isso os torna mais fácil de serem testados e debugados. Também são evitadas duplicações de códigos que poderiam ocorrer, já que o mesmo componente criado pode ser utilizado em diversos outros mantendo apenas sua única estrutura original. Na Listagem 2.3 pode ser visto um exemplo de componente criado com JSX.

Listagem 2.3 – Exemplo de componente em React.

```
1  import React from 'react'
2  import { render } from 'react-dom'
3
4  const App = React.createClass({
5    render() {
6      return (
7        <div>A little component</div>
8      )
9    }
10 })
```

A característica de orientação a componentes do React permite, por exemplo, que uma página possa ser construída com vários pedaços diferentes, que podem ser reutilizados. É muito comum por exemplo, uma aplicação Web possuir diversos botões, e caso essa aplicação tenha sido desenvolvida com React, é provável que todos esses botões sejam replicações de um único componente. O que diferenciaria um botão de outro seriam os argumentos que estes receberiam, como se fossem funções JavaScript. Esses argumentos, para fins de convenção do React, recebem o nome de **props**. Vipul et al. (2016) comenta que as props não podem ser modificadas pelo próprio componente e devem ser tratadas como imutáveis. Caso se deseje alterar propriedades de um componente, deve se alterar o

[1] Documento responsável por representar objetos em formato de árvore: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>.

estado deste, utilizando o conceito de **states**. Neste caso, a cada vez que o estado de um componente for alterado, este será renderizado novamente.

Até 2019, a criação e gerenciamento dos estados das aplicações construídas em React necessitavam de estarem inseridos em uma classe. Porém, a partir de 2019, foi lançado no React a funcionalidade ***hooks*** que, de acordo com Abramov (2019), permite utilizar estado e outras funcionalidades do React sem a necessidade de escrever uma classe. Assim como também é possível criar *hooks* personalizados para compartilhar e reusar a lógica com estado entre componentes. A Listagem 2.4 contém um exemplo de um dos principais *hooks* utilizados, o **useState**. O parâmetro passado a esse *hook* é o valor do estado inicial, e a estrutura do *hook* contém o estado e uma função que altera este estado.

Listagem 2.4 – Exemplo de hooks em React.

```
1  import React, { useState } from 'react';
2
3  function Example() {
4    // Declare uma nova variável de state, a qual chamaremos de "count"
5    const [count, setCount] = useState(0);
6
7    return (
8      <div>
9        <p>Você clicou {count} vezes</p>
10       <button onClick={() => setCount(count + 1)}>
11         Clique aqui
12       </button>
13     </div>
14   );
15 }
```

Neste trabalho, não será utilizado o React em sua versão original, por se tratar de uma versão que é exclusiva para o desenvolvimento Web. Entretanto, cerca de 3 anos após o lançamento do React, o seu criador, o Facebook, publicou o React Native, uma versão do React para dispositivos móveis, que será melhor descrito na Seção 2.3.1.

2.2.4 Firebase

O Firebase consiste em uma plataforma *back-end* focada na construção de aplicações, tanto para a Web, quanto para dispositivos Android e iOS. Adquirida pelo Google em 2014, esta plataforma oferece banco de dados em tempo real, diferentes métodos de autenticação, hospedagem de serviços, entre outros. Em geral, o Firebase oferece certa facilidade aos desenvolvedores, uma vez que com o uso deste não se torna necessário gerenciar o servidor, nem se preocupar com o desenvolvimento de APIs próprias (MEHTA et al., 2017). A plataforma permite que haja foco justamente naquilo que fará a aplicação se tornar um diferencial (MORONEY, 2017).

Diferente do que ocorre em outras aplicações, os dados presentes no banco de dados

do Firebase são armazenados no formato JSON,[2] como descrito por Mehta et al. (2017). Trata-se de um formato de dados flexível e que permite a comunicação de dados entre as diferentes plataformas em tempo real.

O Firebase conta com diversos produtos para atender às diferentes necessidades dos desenvolvedores. Dentre alguns destes, podem ser citados:

- **Firebase Authentication**: destina-se a fornecer uma API de fácil entendimento, que permite construir sistemas de autenticação com um simples login e senha, ou integrá-lo a outros sistemas com o mesmo propósito;
- **Realtime Database**: trata-se de um banco de dados NoSQL hospedado na nuvem. Esse serviço fornece sincronização entre os dispositivos conectados e está disponível mesmo sem conexão com a Internet, por meio de um cache local;
- **Cloud Storage**: permite o armazenamento de outros tipos de dados como fotos e vídeos, por meio de uma API que se comunica e realiza *back-up* com o Google Cloud;
- **Firebase Hosting**: provê hospedagem para aplicações Web, bem como para outros tipos de arquivos, como imagens, áudios, etc.

Segundo Moroney (2017), parte das tecnologias fornecidas pelo Firebase são grátis para uso. Mas mesmo para as tecnologias pagas, é possível fazer seu uso, mas com certas limitações, tornando sua experimentação mais indicada para aplicações menores. Para este trabalho, essa plataforma será suficiente no fornecimento de configurações para o *back-end* da aplicação, bem como na realização de testes.

2.3 Desenvolvimento Multi-Plataforma

Existem diferentes abordagens para o desenvolvimento de aplicações para dispositivos móveis, cada uma com suas vantagens e desvantagens. Silva et al. (2014) define as principais abordagens como sendo:

- **Aplicativos Nativos**: são desenvolvidos para um tipo específico de plataforma, sendo essa plataforma composta de diversas tecnologias, tais como sistema operacional, linguagens de programação e IDEs;
- **Aplicativos Web Puro**: nesta abordagem, faz-se o uso de tecnologias Web (HTML, CSS e JavaScript) para a construção dessas aplicações, que são capazes de apresentar um comportamento similar ao de um aplicativo nativo. Isso é possível devido aos

[2] Um formato de dados JavaScript: <https://www.json.org/json-en.html>.

avanços do HTML5 e CSS3, que incluem funcionalidades essenciais para um aplicativo móvel;

- **Aplicativos Híbridos**: são aplicações Web incorporadas em aplicativos nativos, os quais fornecem uma ponte para o sistema operacional e os serviços nativos do dispositivo. Essa abordagem busca reunir parte das outras abordagens em uma única solução integrada: a flexibilidade de aplicativos Web com a velocidade e riqueza de recursos de aplicativos nativos.

É valido mencionar que, como descrito, os aplicativos nativos são construídos para apenas um sistema operacional. Já as outras duas abordagens possuem realmente a característica de serem "Multi-Plataforma", tema desta seção. Isso permite que estes sejam construídos para mais de um sistema operacional, utilizando apenas um conjunto de tecnologias. Para este trabalho, e após a análise das diferentes abordagens, decidiu-se utilizar o método de desenvolvimento multi-plataforma, devido às suas vantagens em relação às outras que, de acordo com Maharana (2017), são:

- Velocidade de desenvolvimento, devido ao fato de ser possível utilizar uma única base de código para desenvolver aplicações tanto para Android quanto iOS;
- Menor custo de desenvolvimento, pelos mesmos motivos citados acima;
- Simplicidade, uma vez que nessa abordagem é possível atualizar os aplicativos a qualquer momento e consertar bugs sem maiores problemas.

Ainda de acordo com Maharana (2017), as desvantagens dessa mesma abordagem são:

- Experiência do usuário, uma vez que é necessário se atentar às configurações de tela e *layout* dos diferentes sistemas operacionais, procurando garantir a mesma fluidez à ambas;
- Desafios de integração, já que realizar integrações de aplicativos multi-plataforma com recursos nativos dos sistemas operacionais não ocorre de forma direta e pode tornar o código confuso;
- Limitações de plataforma, já que não é possível utilizar conjuntos de funcionalidades fornecidos pelas plataformas. Geralmente se faz uso de *plugins* e bibliotecas fornecidas pelos *frameworks* de desenvolvimento multi-plataforma.

Existem vários *frameworks* que apoiam o desenvolvimento multi-plataforma de aplicativos. Como este trabalho pretende utilizar conceitos Web mencionados na Seção 2.2,

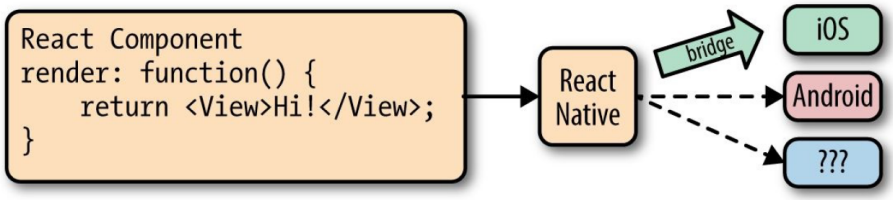

Figura 2 – Funcionamento do React Native (EISENMAN, 2015).

será feito uso do *framework* React Native, que será descrito na próxima seção. Segundo Holmes et al. (2015), através desse *framework* é possível aproveitar habilidades adquiridas com o desenvolvimento Web, tendo acesso a métodos familiares para construção de interfaces em dispositivos móveis.

2.3.1 React Native

De acordo com Eisenman (2015), React Native é um *framework* JavaScript que permite a construção de aplicações reais, que podem ser renderizadas nativamente tanto em ambientes Android quanto iOS. É um *framework* baseado no React, porém ao invés de alvejar navegadores Web, o alvo são as plataformas *mobile*. Em outras palavras: desenvolvedores Web agora podem construir aplicativos móveis que se parecem e agem como nativos, tudo devido ao conforto da biblioteca JavaScript React. Além disso, a maior parte do código escrito pode ser compartilhado entre ambas as plataformas, facilitando o desenvolvimento simultâneo para Android e iOS.

O React Native pode ser considerado um *framework* "híbrido" pelo fato de ser possível construir aplicações por meio deste com apenas uma base de tecnologias, entretanto, este não é considerado totalmente híbrido pelo fato de não fazer o uso de *WebViews* nas aplicações. A técnica utilizada pelo React Native, como descrito por Occhino (2015), consiste em utilizar o JavaScript para renderizar componentes nativos para cada uma das plataformas, estabelecendo certa comunicação direta com as aplicações. Essa técnica poderia sugerir que o fluxo de *threads* e processos na aplicação fossem prejudicados pelas execuções realizadas pelo JavaScript, entretanto isso não ocorre pois a comunicação realizada com as plataformas acontece de forma assíncrona.

A Figura 2, presente em (EISENMAN, 2015), permite uma visualização da ponte que faz a comunicação com as diferentes plataformas. É possível observar que um componente React é traduzido pelo React Native em um componente nativo que de fato existe nas plataformas Android e iOS.

Além disso, essa figura também ilustra uma das principais diferenças existentes entre o React e o React Native: as *tags* utilizadas para criação dos componentes. Basicamente o que muda de uma para outra, é a nomenclatura que se deve usar nas marcações JSX, algo que pode ser percebido ao comparar a Figura 2 com a Listagem 2.3. Enquanto no React

os componentes são criados com as mesmas *tags* utilizadas no HTML, no React Native faz-se o uso de *tags* específicas para cada tipo de componente. A outra principal diferença do React em relação ao React Native está na forma como são estilizados os componentes. De acordo com Eisenman (2015), enquanto o React permite que sejam criados arquivos separados que podem ser utilizados para estilizar componentes, o React Native traz uma abordagem que força o uso da estilização dentro dos componentes criados, de maneira explícita.

De acordo com Holmes et al. (2015), o React Native fornece uma abstração para o mecanismo de armazenamento de dados local, para que não seja necessário entender a diferenças entre os mecanismos de armazenamento utilizados pelas plataformas Android e iOS. Para essa abstração, é necessário a criação de um objeto chamado **AsyncStorage**, o qual será responsável por armazenar e recuperar informações dos dispositivos.

O método utilizado para salvar dados é o **setItem(key: string, value: string)**. Normalmente para a chave se usa o formato **@AppName:key**, portanto, um possível exemplo desse método pode ser descrito na Listagem 2.5.

Listagem 2.5 – Uso do método setItem.

```
1 AsyncStorage.setItem('@Concessionaria:veiculos', 'Fiat Uno')
2 .then ( () => console.log ('Veiculo armazenado com sucesso'))
```

O método utilizado para recuperar informações, é o **getItem(key: string)**. Os dados são recuperados através da chave passada como argumento, como observado no exemplo da Listagem 2.6.

Listagem 2.6 – Uso do método getItem.

```
1 AsyncStorage.getItem('@Concessionaria:veiculos')
2 .then ( data => {
3   if(data != null) {
4     this.setState({vehicles: JSON.parse(data)})
5   }
6 })
```

Existem outros métodos que o AsyncStorage oferece, e podem ser consultados em sua documentação[3] no site oficial.

2.3.2 Expo

De acordo com Ighosewe (2021), o Expo é um conjunto de ferramentas que auxiliam no rápido desenvolvimento de uma aplicação em React Native. Este provê várias ferramentas tanto para o processo de desenvolvimento, quanto para o processo de teste da aplicação.

[3] Documentação oficial AsyncStorage: <https://reactnative.dev/docs/asyncstorage>.

Testar a aplicação, por exemplo, se torna flexível pois existe a possibilidade de se testar tanto no browser, quanto em um dispositivo físico e em emuladores. Além disso, o Expo oferece um SDK que permite que sejam utilizados vários recursos do hardware dos dispositivos, como câmera, mapas, etc.

Uma das ideias principais do Expo é permitir que o desenvolvedor não se preocupe com o código nativo da aplicação, e apenas com os códigos JavaScript/TypeScript. Entretanto, por vezes se torna necessário ter acesso aos códigos nativos e o Expo traz duas abordagens para um projeto. São elas (EXPO, 2022):

- **Managed Workflow**: nessa abordagem apenas é necessário desenvolver a aplicação com códigos JavaScript/TypeScript, e as ferramentas e serviços do Expo lidam com o que ocorre na parte nativa. Não se tem acesso ao código nativo;
- **Bare Workflow**: nessa abordagem é possível utilizar a biblioteca e os serviços do Expo, porém o desenvolvedor também se torna responsável pelos códigos nativos da aplicação, possuindo acesso à estes.

Apesar dos benefícios citado que o Expo provê no desenvolvimento de uma aplicação, este também possui algumas limitações e não é recomendado para aplicações muito complexas, em (LIMITATIONS, 2022) são listadas algumas destas limitações:

- Nem todas as funcionalidades nativas do iOS e Android estão disponíveis;
- A única maneira de utilizar notificações *push* é com a biblioteca do próprio Expo;
- Atualizações e *builds* de aplicações são limitadas por seu tamanho;
- Funcionalidades nativas integradas à bibliotecas externas não estão inclusas no SDK do Expo.

Para este projeto, foi escolhido utilizar a abordagem *Bare Workflow* para a criação do projeto, para o caso de ser necessário realizar modificações nos códigos nativos da aplicação, além de poder acessá-los em outras ferramentas.

3 Especificação de Requisitos

Este capítulo destina-se a apresentar a especificação de requisitos para construção do portal do aluno da UFES na versão *mobile*. Os requisitos foram levantados com base nas funcionalidades presentes na versão Web do portal do aluno da UFES, que são consideradas de grande importância. Além disso, os requisitos também foram levantados baseado nas novas funcionalidades pensadas para esse projeto.

A Seção 3.1 traz a apresentação da descrição do escopo da aplicação implementada. A Seção 3.2 apresenta os requisitos do sistema, contendo as tabelas dos requisitos levantados para esse projeto e o diagrama de casos de uso. Por fim, a Seção 3.3 apresenta o diagrama de classes da aplicação.

3.1 Descrição do escopo

O portal do aluno da UFES na versão Web, é uma plataforma para gerenciar diversas atividades e recuperar documentos importantes para os alunos. Sendo assim, este possui diversas funcionalidades, sendo algumas destas utilizadas com mais frequência do que outras. O escopo do aplicativo referente a esse projeto busca replicar algumas funcionalidades existentes na versão Web do portal do aluno, melhorando a forma como os dados são visualizados quando acessados por um smartphone.

O portal do aluno em uma versão *mobile*, possui capacidades que não são encontrados com tal facilidade na versão Web. A possibilidade de se receber notificações quando atividades importantes estiverem disponíveis, ou quando o número de faltas de alguma disciplina se encontrar em uma situação crítica é algo mais plausível no ambiente *mobile*. Medidas de acessibilidade também estão presentes nesse projeto, como escolha de tema e idioma.

Esse projeto apresenta em sua maior parte, funcionalidades de visualização de dados, como documentos, oferta de disciplinas, calendário acadêmico. Mas também apresenta funcionalidades que originalmente não estão presentes na versão Web do portal do aluno, e foram propostos para esse projeto, como a visualização do saldo/cardápio do Restaurante Universitário, recarga do saldo do Restaurante Universitário, controle de notas e faltas e um *feed* de notícias do colegiado. O projeto também presenta uma seção onde é possível visualizar atividades acadêmicas importantes que estão em curso, como por exemplo a solicitação de matrícula.

Visto que a solicitação para acesso aos dados reais contendo diversas informações dos(as) estudantes da UFES seria complexo demais para o escopo de um trabalho de

conclusão de curso, o projeto é simulado com alguns dados fictícios, bem como dados inseridos manualmente a partir de informações públicas (ex.: disciplinas de cada curso) para efeitos de teste. Não obstante não haver integração com os dados reais da UFES, de fato os dados utilizados pelo sistema desenvolvido estão armazenados em um banco de dados.

Neste projeto, foram utilizadas ferramentas que facilitam a configuração de bancos de dados não relacionais, e que disponibilizam funções para realizar operações neste. Foram criadas então, informações de usuários, que serão, em momento posterior, exibidas na aplicação para visualização dos usuários. Além disso, apesar de algumas das informações que forem exibidas no aplicativo serem de responsabilidade funcional de outros atores, no escopo desse projeto o único ator será o próprio usuário que estiver utilizando o aplicativo.

3.2 Requisitos do sistema

Nesta seção, serão descritos os requisitos levantados para essa aplicação e a modelagem dos casos de uso realizada a partir desses requisitos. A seguir, podem ser visualizadas as tabelas de requisitos funcionais (Tabela 1), regras de negócio (Tabela 2), e requisitos não-funcionais (Tabela 3).

Tabela 1 – Tabela de Requisitos funcionais.

Id	**Descrição**	**Prioridade**	**Dependência**
RF01	O aplicativo deve mostrar ao usuário seu nome e curso após o login.	Alta	
RF02	O aplicativo deve exibir mensagens do mural, permitindo ao usuário decidir o comportamento de exibição.	Alta	
RF03	O aplicativo deve exibir quais atividades acadêmicas periódicas estão disponíveis para acesso.	Alta	
RF04	O aplicativo deve exibir o progresso de disciplinas obrigatórias, optativas e um progresso geral das disciplinas/atividades já concluídas para a graduação.	Média	
RF05	O aplicativo deve permitir que o usuário acesse documentos como Identidade, Ofertas, Relatórios e Calendário acadêmico.	Alta	
RF06	O aplicativo deve exibir mensagens enviadas pelo colegiado.	Alta	
RF07	O aplicativo deve permitir aos usuários reagirem às mensagens enviadas pelo colegiado.	Média	RF06
RF08	O aplicativo deve mostrar o saldo do Restaurante Universitário dos usuários.	Alta	
RF09	O aplicativo deve mostrar o cardápio do dia do Restaurante Universitário.	Alta	
RF10	O aplicativo deve permitir que o usuário recarregue seu saldo do RU.	Alta	RF08
RF11	O aplicativo deve permitir que o usuário configure tema, idioma e recebimento de notificações.	Alta	
RF12	O aplicativo deve permitir que a Oferta de cursos seja filtrada por uma determinada categoria.	Alta	RF05
RF13	O aplicativo deve permitir que o usuário acesse o progresso das disciplinas cursadas no período, contendo média atual e número de faltas de cada disciplina.	Alta	
RF14	O aplicativo deve exibir notificações ao usuário com informações sobre atividades acadêmicas e notícias do colegiado.	Média	
RF15	O aplicativo deve alertar o usuário sobre o risco de reprovação por faltas em determinada disciplina.	Alta	

Tabela 2 – Tabela de Regras de negócio.

Id	**Descrição**	**Prioridade**	**Dependência**
RN01	O valor mínimo de recarga do Restaurante Universitário deve ser igual ao valor mínimo da refeição oferecida por este.	Alta	
RN02	A oferta exibida pelo aplicativo deverá sempre ser do período em curso.	Alta	
RN03	O risco de reprovação por faltas é considerado quando o aluno possui um valor maior ou igual a 75% de faltas.	Alta	
RN04	Inicialmente, os idiomas suportados devem ser português e inglês.	Alta	

Tabela 3 – Tabela de Requisitos não-funcionais.

Id	**Descrição**	**Prioridade**	**Dependência**
RNF01	A aplicação deverá ter uma interface intuitiva e amigável, não necessitando um treinamento prévio para seu uso.	Alta	
RNF02	A aplicação deverá estar disponível para uso tanto nas plataformas iOS quanto Android.	Alta	
RNF03	O tempo de sessão do usuário deverá ser de 10 minutos.	Alta	
RNF04	O sistema deve possuir uma estrutura que permita uma boa manutenabilidade e adição de novas funcionalidades.	Alta	
RNF05	O sistema deve ser desenvolvido incorporando a técnica de reúso de componentes.	Alta	

O portal do aluno possui apenas um subsistema, que está indicado na Tabela 4. O único ator presente no escopo desse projeto é o aluno e, portanto, é o único ator presente no diagrama de casos de uso na Figura 3, que foi modelado a partir dos requisitos apresentados anteriormente. A seguir, serão descritas possibilidades de ações que um usuário terá ao fazer o uso da aplicação considerando a modelagem realizada.

Tabela 4 – Tabela do subsistema da aplicação.

Subsistema	**Descrição**
Core	Envolve todas as funcionalidades da aplicação que podem ser acessadas pelos alunos.

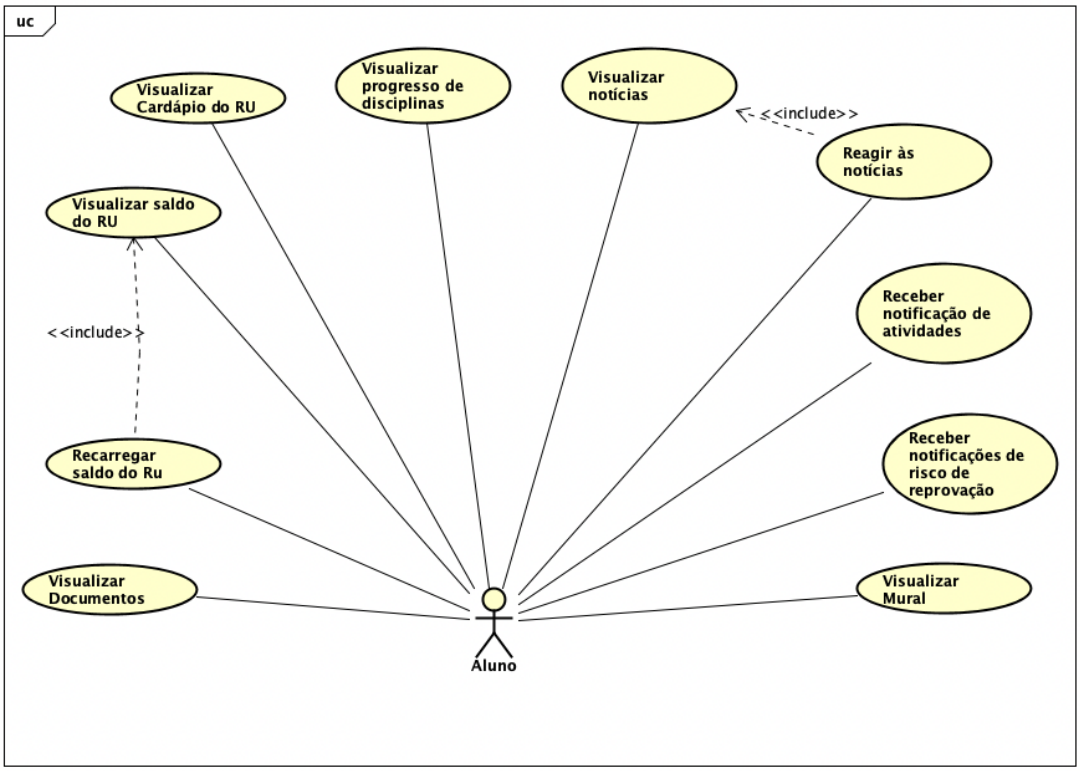

Figura 3 – Diagrama de caso de uso do subsistema core.

Ao efetuar o Login na aplicação, o Aluno estará na tela inicial, onde poderá **Visualizar o Mural**, contendo as informações gerais do portal, que são exibidas a todos os usuários. O usuário terá, a partir da tela inicial, a possibilidade de trocar de aba (tela) e acessar outras funcionalidades.

Um dos possíveis abas é o referente aos documentos do aluno, onde o usuário terá várias opções como Identidade, Histórico de notas, etc. O usuário terá, portanto, a possibilidade de escolher e **Visualizar os documentos** desejados, no formato PDF. Ainda neste aba, também existe a possibilidade do usuário **Visualizar o progresso das disciplinas** que estão sendo realizados por este no período corrente, podendo checar a média atual e o número de faltas em cada disciplina.

Uma outra aba que o aluno poderá navegar é a aba de notícias do colegiado. Trata-se de notícias que são enviadas pelo colegiado de cada curso. Neste aba, além do usuário ter a possibilidade de **Visualizar as notícias**, este também poderá **Reagir às noticias**, indicando que estas possivelmente tiveram um cunho positivo para este.

A última das possíveis abas que o usuário poderá acessar é referente ao Restaurante Universitário. Nesta aba, o usuário terá a possibilidade de **Visualizar o saldo do RU** e, caso o mesmo deseje, poderá ainda **Recarregar o saldo do RU**, utilizando alguma das formas de pagamento disponíveis. Além disso, nesta mesmo aba o usuário terá a possibilidade de **Visualizar o cardápio do RU**, contendo os pratos oferecidos pelo restaurante no dia atual.

Por fim, caso alguma nova atividade acadêmica seja registrada como disponível, o usuário deverá **Receber uma notificação desta atividade**. Da mesma forma, caso alguma disciplina que o usuário esteja realizando for atualizada com um número de faltas considerada de risco, o usuário deverá **Receber uma notificação do risco de reprovação** nesta disciplina.

3.3 Diagrama de classes

A Figura 4 apresenta o diagrama de classes referente à aplicação desse projeto, no escopo definido anteriormente. A seguir, serão apresentadas as classes que aparecem no diagrama.

A classe **Aluno** refere-se ao estudante da Universidade que estará utilizando a aplicação. Trata-se do ator principal do sistema, que fará interações com o aplicativo e, consequentemente, terá relacionamentos com parte das outras classes. A classe **Curso** representa o curso que um determinado aluno está matriculado. E a classe **Disciplina** representa as disciplinas ofertadas pela Univerisdade, que podem pertencer a vários cursos diferentes.

A classe **Documento** representa os documentos (em formato PDF) que o usuário terá a possibilidade de acessar na aplicação. Apesar de todos possuírem o mesmo formato, estes se diferenciam pelo seu tipo, podendo ser ofertas, calendário acadêmico, etc. A classe **ItemMural** representa as mensagens que estão presentes no Mural do portal.

A classe **Atividade academica** representa as atividades acadêmicas em curso sob um determinado período, e o usuário poderá visualizar quais delas estão disponíveis. A classe **Notícia** representa as notícias que são enviadas pelos colegiados dos cursos e podem ser reagidas pelo usuário. Estas reações são representadas pela classe **Curtida**.

A classe **Cardápio RU** refere-se ao cardápio do Restaurante Universitário para uma determinado data, exibindo cada categoria e prato das opções disponíveis, que são

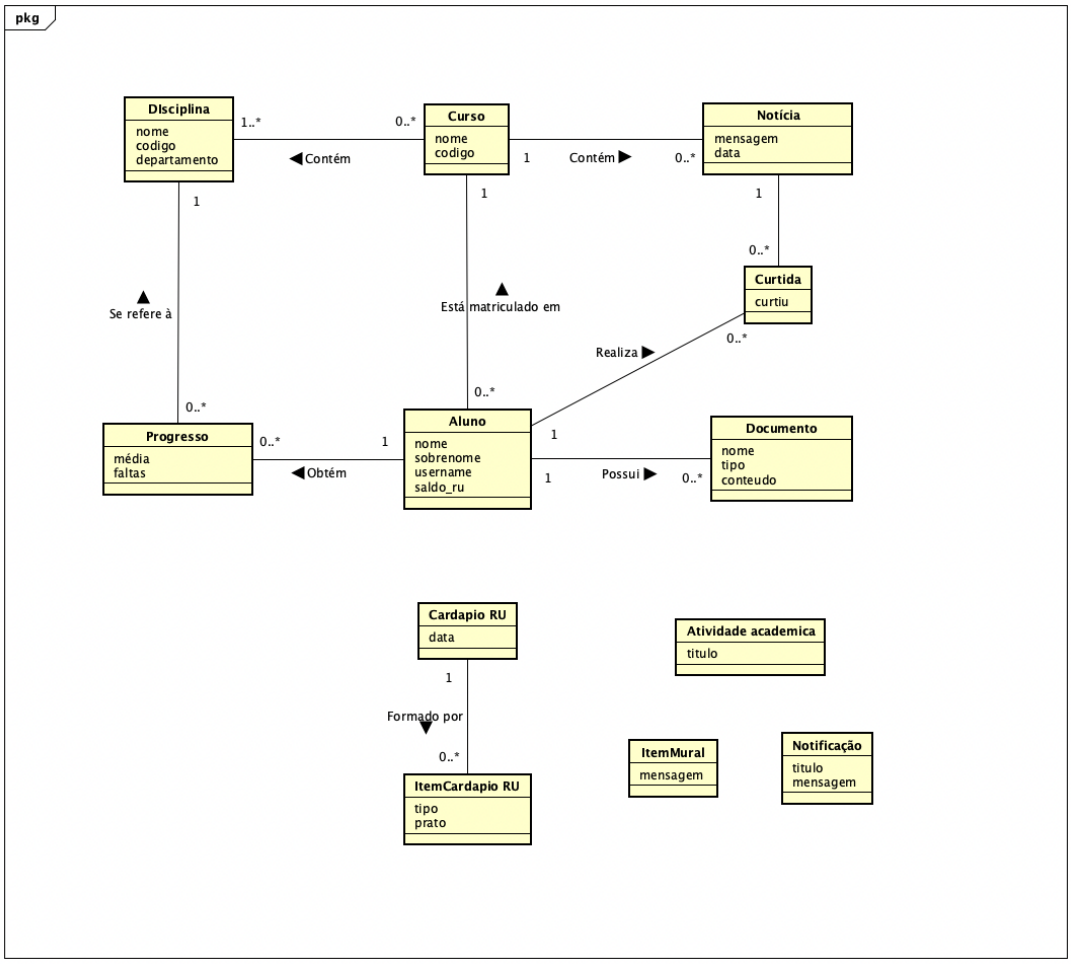

Figura 4 – Diagrama de classes do subsistema core.

representadas pela classe **ItemCardapio RU**.

A classe **Progresso** representa o progresso das disciplinas que estão sendo cursados pelo usuário no período em curso. Cada um destes progressos possui a média atual e número de faltas do aluno. Por fim, a classe **Notificação** representa as notificações que o usuário poderá receber pela aplicação, contendo um título e a mensagem principal.

4 Projeto Arquitetural e Implementação

Nesta seção são discutidos a arquitetura e os detalhes de implementação que foram escolhidos para esse projeto. A Seção 4.1 descreve como está organizada a arquitetura do sistema. Na Seção 4.2 são descritos detalhes da implementação dos vários modelos do sistema. A seção 4.3 contém detalhes da implementação dos módulos específicos do Firebase. Por fim, na Seção 4.4 é apresentado o sistema com imagens de tela capturadas.

4.1 Arquitetura do Projeto

A arquitetura deste projeto é constituída por três camadas. A primeira camada, denominada Camada de Visão (**View**), se refere à tudo na aplicação que pode ser visualizado e interagido pelo usuário. A Camada de Controle (**Controller**), é responsável por mediar informações que são requisitadas e repassá-las entre as outras camadas. E a última camada é denominada Camada de Modelo (**Model**) que possui como responsabilidades realizar operações no banco de dados, e retornar informações requisitadas. A Figura 5 ilustra a arquitetura deste projeto. A seguir detalhamos um pouco mais cada camada.

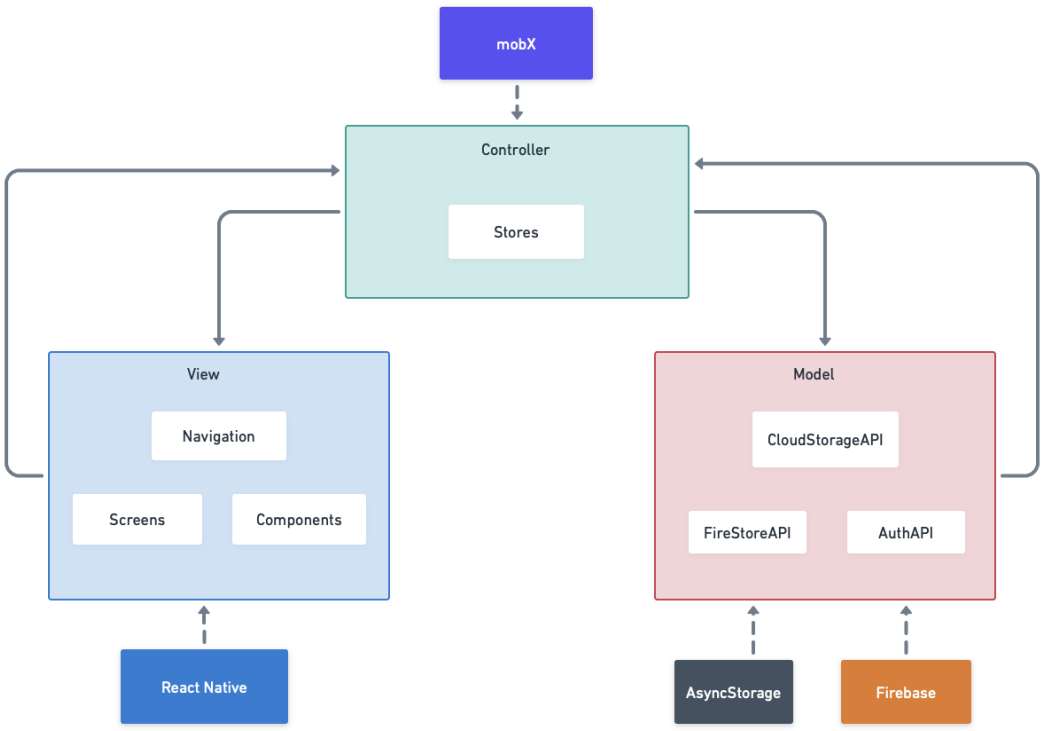

Figura 5 – Arquitetura do sistema.

4.1.1 Camada de Visão

Os componentes que formam a Camada de Visão estão todos reunidos no arquivo **App.tsx**, presente na raiz do projeto. Este é o componente principal da aplicação, que engloba todos os outros componentes que constroem a visualização, interação e navegação da aplicação. Primeiramente é definida a navegação do aplicativo, com os arquivos que irão permitir que o usuário possa alternar entre as telas que o aplicativo oferece. Com a navegação construída, serão adicionadas a esta as telas da aplicação.

As telas da aplicação são construídas tanto com componentes próprios, quanto com os componentes reutilizáveis criados para esta aplicação. Alguns componentes podem até ser considerados uma própria tela da aplicação, como por exemplo o componente PDFViewer, que é uma tela para visualização de documentos PDF.

Toda a criação da navegação da aplicação, telas e componentes utilizou como base tanto funcionalidades nativas do React Native, como várias de suas bibliotecas externas, que facilitaram a implementação desta camada. Além disso, arquivos externos contendo traduções, configurações de cores, imagens e funções que auxiliam em funcionalidades de visualização também fazem parte desta camada.

4.1.2 Camada de Controle

A Camada de Controle está reunida na pasta **stores** do projeto e possui duas classes principais, que são responsáveis pelo controle de dados de diferentes entidades. A classe **UserStore** é responsável pelo controle dos dados referentes a um usuário especifico, que será o usuário que acessará a aplicação. Já a classe **InfoStore** é responsável pelo controle dos dados gerais da aplicação, não sendo necessariamente referentes a algum usuário.

As classes de controle atuam com dois papeis principais, o primeiro deles é servir como uma ponte entre as camadas de Visão e Modelo, repassando requisições e devolvendo as respostas desejadas para as camadas. Para que um documento PDF seja exibido na aplicação, por exemplo, a Camada de Visão irá realizar uma requisição à classe controladora UserStore, que fará uma outra requisição à Camada de Modelo. Assim que esta última camada retornar o endereço referente ao documento em questão, a classe controladora irá repassá-lo à Camada de Visão.

O segundo papel das classes controladoras é o controle dos estados globais da aplicação, o qual é feito com o uso de método específicos presentes na classe, que são fornecidos pela biblioteca mobX. Esta biblioteca define a nomenclatura das classes como Stores, portanto, foi mantida essa nomenclatura nas classes de controle implementadas, apesar destas também possuírem um outro papel mencionado no parágrafo anterior.

4.1.3 Camada de Modelo

A camada de Modelo está reunida na pasta **api** e contém os métodos que farão operações nos banco de dados, separados por módulos. Vale ressaltar que esta aplicação faz uso do banco de dados tanto dos módulos do Firebase, quanto do armazenamento interno do dispositivo.

O módulo **AuthAPI** contém tanto métodos que realizam operações de autenticação do usuário no banco do Firebase, quanto métodos que realizam operações no armazenamento interno do dispositivo físico com o uso do AsyncStorage. Este é o único módulo que realiza operações de escrita e leitura no banco de dados, os outros módulos realizam apenas operações de leitura (consultas).

O módulo **FireStoreAPI** contém os métodos que realizam consultas em informações presentes no banco de dados Firestore do Firebase. Nestas informações estão contidas tanto informações gerais da aplicação, quanto todos os dados necessários dos usuários para a aplicação, com exceção dos documentos PDF, pois estes são consultados pela aplicação através do módulo **CloudStorageAPI**, que é responsável apenas por retornar os endereços dos documentos para que a aplicação os possa exibir.

4.2 Detalhes de Implementação

Esta seção traz detalhes dos principais tópicos que compuseram o processo de desenvolvimento desta aplicação, com exceção dos tópicos que foram desenvolvidos com o Firebase, que serão detalhados na Seção 4.3.

4.2.1 Organização de pastas

A organização de pastas deste projeto procurou encapsular tudo o que realmente foi produzido em uma única pasta, e deixá-la separada de outras pastas e arquivos que são gerados automaticamente pelo Expo. Na Figura 6 é possível ver a organização geral de pastas e arquivos na raíz do projeto.

A pasta **.expo** contém arquivos de configuração específicos do Expo, que são utilizados para configurar alguns de seus dados. As pastas **android** e **ios** contém os projetos que são gerados para cada um dos sistemas operacionais. A pasta **node_modules** contém todas as bibliotecas JavaScript necessárias para o funcionamento do projeto. A pasta **src** contém todas as pastas e arquivos que de fato foram produzidos para esse projeto. Por fim, dos arquivos que estão na raíz do projeto, os mais importantes a serem citados são o **App.tsx** que é o componente que engloba toda a aplicação, o **package.json** que contém a lista com todas as bibliotecas JavaScript e suas versões utilizadas, e o arquivo **tsconfig.json** que contém informações a respeito das regras do TypeScript utilizadas.

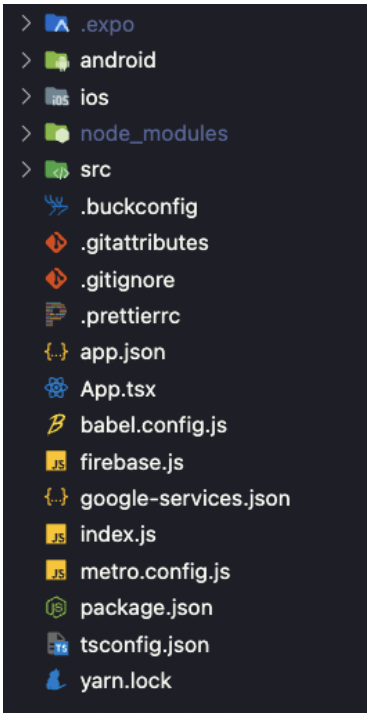

Figura 6 – Organização de pastas do sistema.

A estrutura da pasta **src**, exibida na Figura 7, contem as seguintes pastas:

- **api**: contém os arquivos responsáveis por fazer a ponte com a API do Firebase e retornar os dados requisitados;
- **assets**: contém todos os arquivos de imagens e ícones que foram utilizados pelas diversas telas do projeto;
- **components**: contém todos os componentes que foram construídos pela aplicação e que foram reutilizados em diversas telas;
- **hooks**: contém os *hooks* personalizados que foram criados para controle de estados;
- **locale**: contém os arquivos de internacionalização, com as traduções para português e inglês;
- **navigation**: contém os componentes responsáveis por permitir a navegação entre telas na aplicação;
- **screens**: contém todas as telas que foram construídas e que podem ser exibidas na aplicação;
- **stores**: contém as classes controladoras do aplicativo, desenvolvidas com o mobX (vide Subseção 4.2.4);
- **styles**: contém as definições do tema da aplicação;

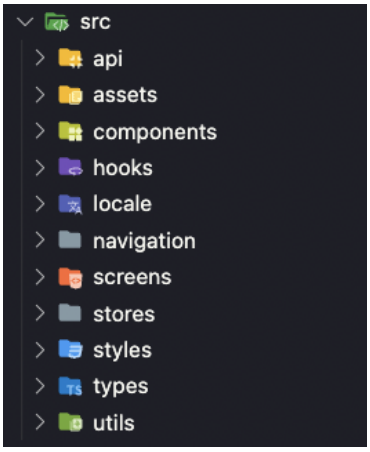

Figura 7 – Organização da pasta src.

- **types**: contém as definições de tipo globais do TypeScript;
- **utils**: contém diversos métodos que auxiliaram no desenvolvimento de funcionalidades.

4.2.2 Navegação

As funcionalidades de navegação de telas foram desenvolvidas utilizando-se a biblioteca externa **react-navigation**.[1] Existem dois tipos de navegação que foram utilizados na aplicação, a navegação por pilhas é utilizada quando se deseja navegar entre as telas e ter um rastreamento das que foram visitadas, para possibilitar retorno às telas anteriores. Cada um dos componentes de tela então é empilhado conforme se navega. Na Listagem 4.1 é possível observar o componente que determina a navegação por pilha da aplicação.

A criação da pilha de navegação é feita pelo método **createStackNavigator()** e todas as telas que farão parte dessa pilha devem ser listadas dentro do componente criado por esse método.

O outro tipo de navegação presente na aplicação é a navegação por abas, onde nesse caso será criado um componente de navegação contendo referências (botões) para as telas que se pode alternar. Mas as telas são independentes entre si e a navegação entre elas na verdade é uma substituição da renderização de uma tela por outra. A codificação da criação do componente (Listagem 4.2) é bem parecida com o tipo de navegação por pilha, mas utiliza o método **createBottomTabNavigator()** para criação do componente, e requer parâmetros adicionais para cada tela, como por exemplo o ícone que fará referência àquela tela.

É interessante observar que a navegação por abas está contida dentro da navegação por pilhas, ou seja, uma das pilhas de navegação é o próprio componente de navegação

[1] React-Navigation: <https://reactnavigation.org>.

Listagem 4.1 – Componente de navegação em pilhas.

```
export function StackNavigator() {
    const Stack = createStackNavigator();
    return(
        <NavigationContainer theme={{ colors: { background: '#000' } }}>
            <Stack.Navigator
              initialRouteName="Login"
              screenOptions={{
                cardStyle: { backgroundColor: 'transparent' },
                ...TransitionPresets.SlideFromRightIOS
              }}
            >
              <Stack.Screen
                name="Login"
                component={Login}
                options={{ headerShown: false }}
              />
              <Stack.Screen
                name="Routes"
                component={TabNavigator}
                options={{ headerShown: false }}
              />
            // ... Outras Telas
            </Stack.Navigator>
        </NavigationContainer>
    );
}
```

Listagem 4.2 – Componente de navegação em abas.

```
export function TabNavigator() {
  const [,setUpdate] = useState(false);
  const { Navigator, Screen } = createBottomTabNavigator();
  return (
    <Navigator
      screenOptions={{
        headerShown: false,
        tabBarActiveTintColor: theme.colors[getTheme()].select,
        tabBarInactiveTintColor: 'grey',
        tabBarStyle: {
          backgroundColor: theme.colors[getTheme()].background_secondary,
          borderTopWidth: 0
        }
      }}
    >
      <Screen
        name={locale('general.home')}
        component={Home}
        options={{
          tabBarIcon: ({ size, color }) => (
            <MaterialIcons name="home" size={size} color={color} />
          )
        }}
      />
      // ... Outras telas
    </Navigator>
  );
}
```

por abas. Isso permite que sejam utilizadas os dois tipos de navegação ao mesmo tempo, trazendo uma boa dinâmica ao aplicativo.

4.2.3 Criação de componentes e estilização

A criação dos componentes desta aplicação, sejam os reutilizáveis ou as telas, utilizam a biblioteca **styled-components**[2] como base para sua estrutura e estilização. Em relação à criação dos componentes, essa biblioteca permite que estes sejam criados com nomes personalizados, ao invés de se utilizar as estruturas padrão do React Native (View, Text, etc..). E isso permite uma maior legibilidade ao código, visto que o componente pode possuir um nome que faz referência direta à sua característica.

Em relação à estilização, a biblioteca permite que se atribua a um componente, qualquer tipo, seja de outro componente personalizado ou primitivo. Além disso, é possível estilizar os componentes utilizando a mesma sintaxe do CSS, pois os códigos escritos em CSS são convertidos para o StyleSheet do React Native.

A Listagem 4.3 contém o exemplo de um componente criado para ser utilizado em formulários. É possível perceber que cada componente que o compõe possui um nome personalizado, que foi definido em outro arquivo, contendo suas respectivas definições e estilizações.

Listagem 4.3 – Utilização de um componente com styled-components.

```
export function InputForm({
    label,
    placeholder,
    value,
    error,
    handleChange,
    autoCapitalize,
    secureTextEntry,
    ...props}:Props & TextInputProps) {

    return(
        <Container {...props}>
            <Label>{label}</Label>
            <InputText
                onChangeText={handleChange}
                value={value}
                error={error}
                placeholder={placeholder}
                autoCapitalize={autoCapitalize}
                secureTextEntry={secureTextEntry}
                autoCorrect={false}
            />
            <Error>{error}</Error>
        </Container>
    )
}
```

A definição e estilização dos componentes presentes no componente citado na

[2] styled-components: <https://styled-components.com>.

listagem anterior estão presentes na Listagem 4.4. É possível perceber que a definição do tipo do componente é feito através do objeto **styled**, presente na biblioteca. E a estilização é definida dentro de uma string (no formato de Template string[3]) que está acoplada a esse objeto.

Listagem 4.4 – Definição e estilização de um componente com styled-components.

```
1  export const Container = styled.View`
2      margin-top: 20px;
3  `;
4
5  export const Label = styled.Text`
6      color: ${({theme}) => theme.colors[getTheme()].label_input}
7  `;
8
9  export const InputText = styled(TextInput).attrs({placeholderTextColor: '#9B9B9D
       '})`
10     background-color: ${({theme}) => theme.colors[getTheme()].background_input};
11     border-radius: 5px;
12     margin-top: 10px;
13     height: ${RFValue(45)}px;
14     padding-left: 10px;
15     color: ${({theme}) => theme.colors[getTheme()].main_text};;
16 `;
17
18 export const Error = styled.Text`
19     color: #D95858;
20     font-size: 12px;
21 `;
```

4.2.4 Gerenciamento de estados

Estados são variáveis que, quando mudam de valor, devem sofrer uma nova renderização nos componentes em que estiverem inseridas. Quando o escopo desses estados é dentro de um único componente, é simples gerenciar o estado utilizando o *hook* **useState** do React. Porém quando existem estados que são compartilhados por toda a aplicação, se faz necessário o uso de técnicas mais complexas para gerenciar esses estados globais.

O gerenciamento de estados globais da aplicação foi implementado com o auxílio da biblioteca **mobX**[4]. Basicamente são criadas classes controladoras (o mobX as denomina *stores*), que em um primeiro momento serão apenas classes criadas com recursos da linguagem JavaScript. Porém, os atributos e métodos destas classes podem receber um rótulo específico, indicando quais serão suas respectivas funções nessa classe.

Neste projeto, as classes controladoras (*stores*) foram criadas tanto para controle de estados, quanto para servir de ponte à API do Firebase. Os métodos que são utilizadas para controle de estados, recebem o decorador **@action** e os estados definidos na classe recebem o decorador **@observable**.

[3] Template string: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Template_literals>.

[4] mobX: <https://mobx.js.org>

Na Listagem 4.5 é possível observar a criação da classe controladora do Usuário. Os estados são definidos na parte superior do código, onde também poderiam ter sidos definidos atributos normais, que não estados. E logo abaixo estão presentes tanto os métodos responsáveis por alterar os estados, quanto os demais métodos.

Listagem 4.5 – Classe controladora do Usuário.

```
export default class UserStore {

    @observable
    userInfo = {} as UserInfoProps;

    @observable
    userName = '';

    @observable
    userSubjects = [];

    @action
    login = async(email:string, password:string) => {
        await AuthApi.login(email, password);
        this.userName = email;
    };

    @action
    getUserInfo = async() => {
        const data = await FireStoreApi.getUserInfo(this.userName);
        this.userInfo = data as UserInfoProps;
    };

    getUserCourseSubjects = async() => {
        const {course} = this.userInfo;
        const data = await FireStoreApi.getUserCourseSubjects(course.code);
        return data;
    }
}
```

Com a ajuda dos *hooks* do React-Native, foi possível criar um *hook* chamado **useStores**, para facilitar o acesso aos atributos das *stores*. A Listagem 4.6 mostra um exemplo onde é possível recuperar as classes de controle, e fazer uso de seus métodos e/ou acessar seus estados onde for necessário.

4.2.5 Responsividade

A responsividade das telas da aplicação, isto é, a proporção de dimensões e espaçamentos dos componentes aos diferentes tamanhos de tela, foram implementadas fazendo-se o uso de duas bibliotecas. A primeira delas é a **react-native-responsive-fontsize**[5] que, como o nome sugere, procura trazer uma funcionalidade que busca ajustar os tamanhos de fontes proporcionalmente à altura dos dispositivos, e seu método principal é o **RFValue()**. Essa funcionalidade embora ser em um primeiro momento sugerida para ajustar tamanhos de fontes, também pode ser aplicada a ajustes de espaçamentos situados na vertical.

[5] react-native-responsive-fontsize: <https://github.com/heyman333/react-native-responsive-fontsize>

Listagem 4.6 – Exemplo do hook useStores.

```
const Login = ({navigation}) => {
  const { user, info } = useStores();
  const [loaded, setLoaded] = useState(false);

  const onSubmit = async(values: FormLogin) => {
    try {
      setLoaded(true);
      await user.login(values.email, values.password);
      await Promise.all([user.getUserInfo(), info.getGeneralInfo()]);
      navigation.dispatch(CommonActions.reset({
        routes: [
          { name: 'Routes' },
        ],
      }));
      setLoaded(false);
    } catch (err) {
      showError(locale(`firebase.${err.code}`));
      setLoaded(false);
    }
  };

  // ... resto do código

}
```

A segunda biblioteca utilizada para fins de responsividade, foi a **react-native-responsive-screen**.[6] Essa biblioteca provê dois métodos que convertem medidas passadas como parâmetros para a densidade de pixel referente ao dispositivo. O método **widthPercentageToDP()** realiza essa conversão baseada no comprimento do dispositivo, enquanto que o método **heightPercentageToDP()** faz essa conversão considerando a altura do mesmo. Essa biblioteca é bastante útil caso seja necessário que um componente ocupe o mesmo espaço seja vertical ou horizontal, em qualquer dispositivo.

Apesar das bibliotecas possuírem objetivos e cálculos diferentes em seus métodos, as funcionalidades de ambas foram utilizadas para definir tamanhos de fontes, espaçamento e dimensão dos componentes. Em cada caso foram testados os métodos e selecionados aqueles que produziram o melhor resultado. A Listagem 4.7 ilustra essa alternância de funcionalidades na estilização de uma das telas da aplicação.

Na Seção 4.4 será mostrado um exemplo de dois dispositivos com dimensões de telas diferentes, mas que mantiveram a proporção dos componentes fazendo-se o uso das bibliotecas mostradas nessa seção.

4.2.6 Tipagem de dados

A tipagem de todos os tipos de dados foram em sua maioria feitos dentro dos componentes que os utilizaram. Isso ocorreu pois não foram identificados tipos que se repetissem entre diferentes componentes. Caso isso ocorresse, a criação destes tipos teria

[6] react-native-responsive-screen: <https://github.com/marudy/react-native-responsive-screen>

Listagem 4.7 – Uso das funcionalidades de responsividade.

```
import { RFValue } from 'react-native-responsive-fontsize';
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from 'react-
    native-responsive-screen';

export const Container = styled.View`
    flex: 1;
    background-color: ${ ({theme}) => theme.colors[getTheme()].
        background_secondary};
`;

export const Title = styled.Text`
    font-size: ${hp(5.5)}px;
    font-family: ${({theme}) => theme.fonts.primary_400};
    color: ${ ({theme}) => theme.colors[getTheme()].main_text};
    margin-top: ${RFValue(60) + getBottomSpace()/2}px;
    margin-bottom: ${RFValue(20)}px;
`;
```

sido feita em um arquivo separado para termos de reusabilidade.

O TypeScript permite diferentes formas de se criar tipos, podem ser declarados utilizando **interface** ou **type**. Para essa aplicação, na criação de tipos simples de dados, foi utilizada a primeira abordagem, pois esta possui uma sintaxe mais simples e legível. Porém existem tipos de dados que são mais complexos que requerem, por exemplo, união de tipos de dados. Esse tipo de operação está disponível apenas para a segunda abordagem, que também foi utilizada na aplicação.

A Listagem 4.8 mostra um exemplo onde foram definidos tipos utilizando-se interface, bem como sua atribuição a um estado da classe controladora. A Listagem 4.9 contém um exemplo onde foi necessário utilizar **type**, pois foi necessário mesclar duas tipagens diferentes, operação só permitida com esse tipo de abordagem.

4.2.7 Tema e internacionalização

Os assuntos desta seção estão reunidos pois a implementações de suas funcionalidades apresentam grandes semelhanças. Em resumo, foram definidos arquivos com possíveis valores tanto para o tema, quanto para a internacionalização da aplicação. Após isso, foram implementadas funcionalidades que resgatam esses valores baseados em condições específicas.

Para a internacionalização, como a aplicação inicialmente suporta os idioma português e inglês, foram criados arquivos que contém traduções para todos os textos presentes nas telas. Esses textos foram divididos em várias categorias e sub-categorias, que facilitam sua recuperação através do método **locale()**. Esse método foi implementado para receber como parâmetro alguma categoria e sub-categoria, e retornar o respectivo valor do texto, baseado no idioma atual da aplicação. A Listagem 4.10 contém uma parte do arquivo de traduções e a Listagem 4.11 contém um exemplo de um componente que utilizou o método

Listagem 4.8 – Uso de tipagem com interface.

```
interface UserInfoProps {
    username: string;
    firstname: string;
    lastName: string;
    ru_balance: number;
    course: CourseProps;
}

interface CourseProps {
    code: number;
    name: string;
}

export default class UserStore {

    @observable
    userInfo = {} as UserInfoProps;

    @observable
    userName = '';

    // ... resto do código

}
```

Listagem 4.9 – Uso de tipagem com type.

```
interface ExtraProps {
  title: string;
  maxHeight: number;
};

type Props = DropDownPickerProps & ExtraProps;

export function DropDown({ title, maxHeight, ...props }: Props) {
  // ... Definição do componente
}
```

locale para resgatar um valor de texto.

O arquivo que define o tema da aplicação possui os possíveis valores de cores para os temas claro e escuro, e também define as possíveis fontes que podem ser utilizadas. Nem todas as categorias de cores que estão disponíveis no tema escuro estão presentes no tema claro e vice-versa. Isso ocorre pois cada um desses temas possui suas próprias características, portanto possuem especificidades particulares. A Listagem 4.12 contém as definições mencionadas neste parágrafo.

Os valores de tema da aplicação foram passados a todos os componentes utilizando-se o **ThemeProvider**, componente provido pela biblioteca **styled-components**, mencionada na Subseção 4.2.3. Portanto, todo componente recebe como parâmetro um objeto contendo o tema da aplicação. Para complementar, foi criado o método **getTheme()**, que retorna se a aplicação está utilizando o tema claro ou escuro. A Listagem 4.13 contém um exemplo de estilização que acessa o tema da aplicação utilizando as props do React Native e o

Listagem 4.10 – Arquivo de traduções em inglês.

```
{
    "general": {
        "greetings": "Hello",
        "course": "Course",
        "copy": "Copy",
        "back": "Back",
        "generalLabel": "General",
        "mandatoryLabel": "Mandatory",
        "optionalLabel": "Optional",
        "home": "Home",
        "documents": "Documents",
        "feed": "News",
        "subjects": "Subjects",
        "ru": "UR",
        "viewPDF": "View PDF"
    },

    // ... resto das traduções
}
```

Listagem 4.11 – Uso do método locale.

```
<Title>{`${locale('general.greetings')}, ${userInfo.firstname}`}</Title>
```

Listagem 4.12 – Arquivo de definições de tema.

```
export default {
  colors: {
    dark: {
      background_primary: '#2c2f33',
      background_secondary: '#1C1C1C',
      main_green: '#34AA71',
      select: '#7289da',
      // ... resto das cores
    },
    light: {
      background_primary: '#F5F5F5',
      background_secondary: '#FEFEFE',
      main_dark: '#3E4248',
      main_green: '#1D8E3E',
      select: '#1D8E3E',
      // ... resto das cores
    }
  },
  fonts: {
    primary_400: 'Poppins_400Regular',
    primary_500: 'Poppins_500Medium'
  }
}
```

método mencionado neste parágrafo.

Listagem 4.13 – Uso do tema na estilização.

```
export const Label = styled.Text`
  color: ${({theme}) => theme.colors[getTheme()].main_text};
  margin-bottom: ${hp(1.5)}px;
  font-size: ${hp(2.4)}px;
`;
```

4.2.8 Controle de sessão

O controle de sessão permite que o usuário fique logado por um determinado tempo no aplicativo sem a necessidade de fazer login novamente caso este seja fechado ou reiniciado. Para essa funcionalidade, foi necessário utilizar o **AsyncStorage** para armazenamento e recuperação de dados no dispositivo, auxiliado pela biblioteca **@react-native-async-storage/async-storage**.[7] Também foi necessário utilizar um meio para possibilitar a criação e autenticação de um *token* para os usuários. Para este fim, foi utilizada a biblioteca **expo-jwt**.[8]

O fluxo do controle de sessão consiste em verificar, no momento que o usuário abre o aplicativo, se este possui um *token* armazenado em seu dispositivo e se esse *token* é válido. Caso seja um *token* válido, a aplicação irá redirecionar o usuário para a página principal, mas se o *token* não for válido, o usuário ficará na tela de Login. Além disso, ao fazer Login, será atribuído imediatamente um *token* válido para o usuário. A Listagem 4.14 contém o código da funcionalidade de atribuição de *token*, onde o método **encode()** cria um *token* contendo a data atual e o nome do usuário, além de uma chave secreta que é utilizada para validação.

Listagem 4.14 – Atribuição de um token válido ao usuário.

```
export const setUserToken = async(username: string) => {
  try {
    const token = JWT.encode({exp: Date.now(), user: username}, 'secret-mobile-
        ufes');
    await AsyncStorage.setItem('@storage_token', token);
  } catch (error) {
    throw new ResponseError(error);
  }
}
```

A Listagem 4.15 contém o método responsável por verificar se o usuário tem um *token* válido. Através do método **decode()**, caso a chave secreta esteja correta, será possível obter informações como data de expiração e nome do usuário. Serão verificados então,

[7] @react-native-async-storage/async-storage: <https://github.com/react-native-async-storage/async-storage#readme>

[8] expo-jwt: <https://github.com/blake-simpson/expo-jwt#readme>

quantos minutos de diferença existem entre a data de expiração e a data atual, caso seja um valor menor que o definido nas regras de negócio, será considerado um *token* válido, caso contrário será invalidado e o usuário terá que fazer Login novamente.

Listagem 4.15 – Verificação de token.

```
export const hasValidToken = async() => {
  let result = false;
  try {
    const storedToken = await AsyncStorage.getItem('@storage_token');
    if(!storedToken)
      result = false;

    const token = JWT.decode(storedToken, 'secret-mobile-ufes');
    const expDate = token.exp;
    if((Date.now() - expDate)/60000 < 10) {
      result = true;
    }

    return {result, username: result ? token.user : ''};
  } catch (error) {
    throw new ResponseError(error);
  }
}
```

Por fim, caso o usuário realize Logout na aplicação, o *token* armazenado no dispositivo será substituído por um *token* inválido e portanto, o usuário só conseguirá acesso novamente realizando Login.

4.3 Firebase

Nesta seção, serão mostradas as tecnologias do Firebase que foram utilizadas nesta aplicação, bem como detalhes de suas implementações.

4.3.1 Configuração e gerenciamento

A criação do projeto no Firebase é bem simples, e requer apenas que o usuário forneça dados como nome do projeto, sistema operacional, entre outros. Após todos os dados necessários serem fornecidos, a plataforma irá prover credenciais necessárias para se conectar com a aplicação.

A biblioteca utilizada na aplicação para se conectar e acessar seus módulos é a **firebase**.[9] Essa biblioteca é bem modularizada e permitir acessar as funcionalidades apenas dos módulos desejados. A conexão do Firebase com a aplicação foi realizada através do método **initializeApp()**, cujos parâmetros são justamente as credenciais fornecidas pela plataforma. A Listagem 4.16 contém (com dados fictícios) a forma como a conexão com o firebase foi feita na aplicação.

[9] firebase: <https://firebase.google.com/docs/web/setup>

Listagem 4.16 – Conexão com o Firebase.

```
const firebaseConfig = {
  apiKey: "IaSy2TJ6gkmWAxZouZKiAXTkgdL",
  authDomain: "ufes-mobile-b43243",
  projectId: "ufes-mobile-db342",
  storageBucket: "ufes-mobile-231231.appspot.com",
  messagingSenderId: "48593084583045",
  appId: "1:45345345345345:web:45345345345345b3asdb"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

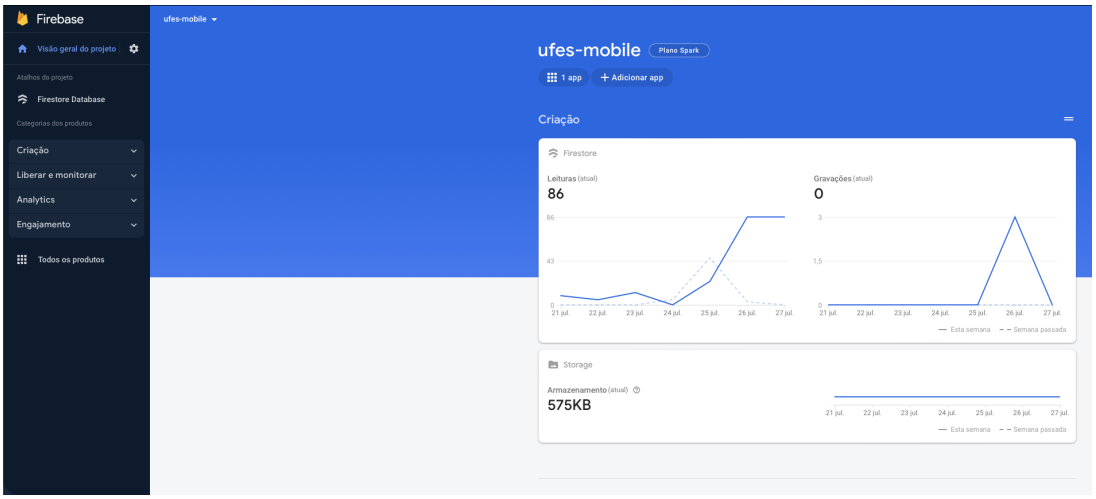

Figura 8 – Console do Firebase.

O gerenciamento dos dados no Firebase foram feitos pelo Console, uma plataforma completa que permite que operações sejam feitas no banco de dados de maneira simples. Existem diversas outras funcionalidades disponíveis no Console, porém esse projeto apenas fez o uso de sua interface gráfica para operações no banco de dados. A Figura 8 ilustra a página inicial do console do Firebase quando é acessado um determinado projeto.

4.3.2 Authentication

O módulo de autenticação do Firebase provê uma base de dados específica que servirá para outras aplicações poderem verificar se determinados usuários estão cadastrados e/ou validados. Essa base de dados conterá apenas usuários com suas respectivas informações e o provedor de login atrelado à estes. O provedor de login utilizado nesta aplicação possui como dados necessários para autenticação um e-mail e uma senha, ou seja, todos os usuários cadastrados deverão possuir esses dados. A Figura 9 mostra o módulo de autenticação no console do Firebase, onde é possível listar, adicionar, e editar informações dos usuários.

É importante ressaltar que na aplicação deste projeto não é necessário escrever um e-mail no campo do nome de usuário para fazer login, pois não é assim que originalmente

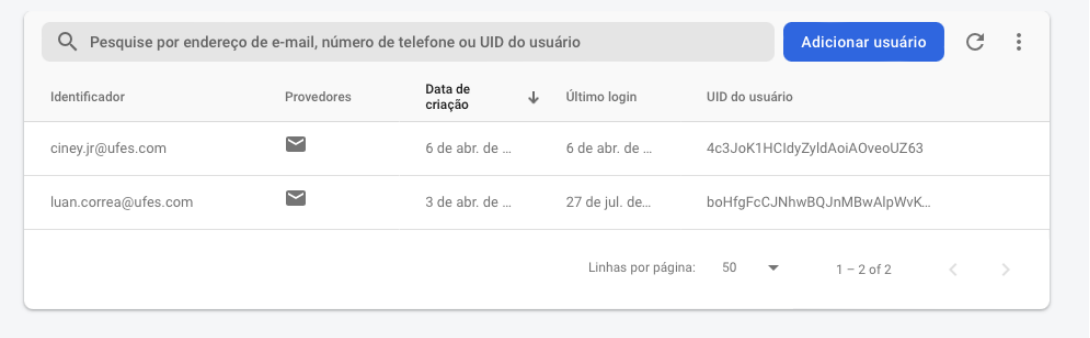

Figura 9 – Gerenciamento de autenticação no Firebase.

é feito no Portal do Aluno da UFES. Porém o Firebase não possui um provedor de login que permita por exemplo um nome de usuário personalizado e uma senha. Portanto, para contornar essa limitação, foi concatenado um e-mail fictício a todos os nomes de usuário e selecionado o provedor de e-mail e senha como provedor de login.

A Listagem 4.17 contém o método de login da aplicação, que foi o único a utilizar as funcionalidades de autenticação do Firebase. A chamada do método **signInWithEmailAndPassword()** provido pela biblioteca, é o que determinará se um usuário está autenticado ou não. Caso nenhum erro seja lançado pelo método significa que o usuário está autenticado, caso contrário, o usuário pode não existir na base de dados ou não estar validado. Ainda nesta listagem também é possível observar como o problema de provedor de login citado no parágrafo anterior foi contornado.

Listagem 4.17 – Login com autenticação do Firebase.

```
export const login = async (username, password): Promise<void> => {
    try {
        const auth = getAuth();
        const userNameFormatted = `${username.trim()}@ufes.com`;
        await signInWithEmailAndPassword(auth, userNameFormatted, password.trim()
            )
    } catch (error) {
      throw new ResponseError(error);
    }
};
```

4.3.3 Firestore

O módulo do Firestore foi utilizado para criação de todos os dados do usuários que não são referentes à autenticação. Também foram utilizadas funcionalidades providas pela biblioteca do Firebase para resgatar estes dados na aplicação. A criação dos dados quando feita pelo console do Firebase se torna bem simples devido à sua interface amigável e intuitiva, e pode ser observada na Figura 10.

As funcionalidades de recuperação dos dados implementadas na aplicação foram consultas sem um alto nível de complexidade, porém, ainda assim foram necessários

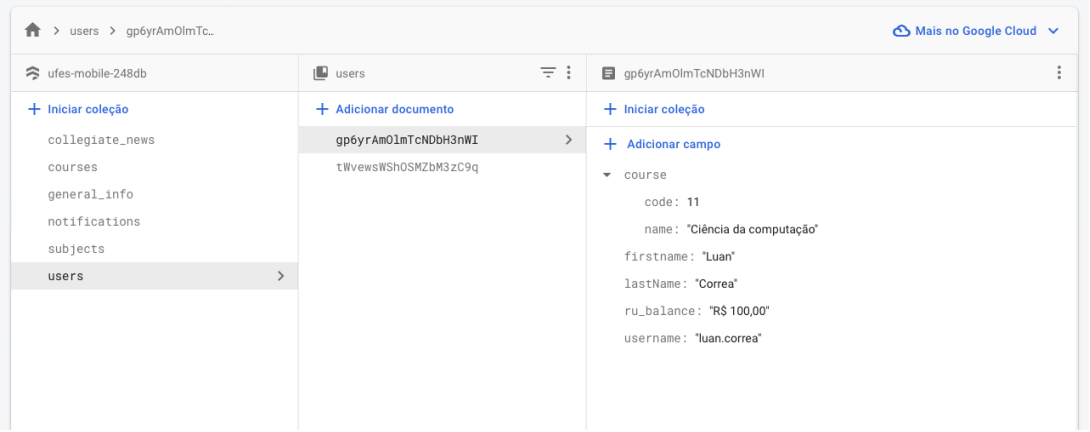

Figura 10 – Gerenciamento de dados no firestore.

vários métodos do módulo do firestore para realizar as consultas, e são eles: **collection()**, **query()**, **where()**, **getFirestore()**, **getDocs()**. A Listagem 4.18 contém um exemplo de um método que realiza uma consulta das notícias de um colegiado baseado no curso do usuário.

Listagem 4.18 – Exemplo de consulta com o Firestore.

```
export const getCollegiateNews = async (courseId: number) => {
  try {
    const db = getFirestore();
    const usersRef = collection(db, "collegiate_news");
    const q = query(usersRef, where("course_id", "==", courseId));
    const result = await getDocs(q);
    return result.docs.shift().data();
  } catch(error) {
    throw new ResponseError(error);
  }
}
```

Uma outra funcionalidade do Firestore foi utilizada para identificar mudanças no banco de dados e disparar, através de outras bibliotecas, notificações para o usuário. Essa funcionalidade será melhor discutida na Seção 4.3.5.

4.3.4 Cloud Storage

O módulo do Cloud Storage foi utilizado como um servidor para hospedar os documentos (em formato PDF) que podem ser acessados pelos usuários. Cada usuário possui uma pasta, sendo que o nome da pasta é seu nome de usuário, e essa pasta contém os documentos que foram solicitados o acesso na aplicação. A Figura 11 contém um exemplo, no console do Firebase, de uma pasta de um usuário com alguns documentos.

Cada um dos documentos armazenados no Cloud Storage possui uma URL associada a este. São esses endereços que são recuperados pela aplicação para fazer a exibição na mesma. Fez-se necessário o uso dos métodos **getStorage()**, **ref()** e **getDownloadURL()**

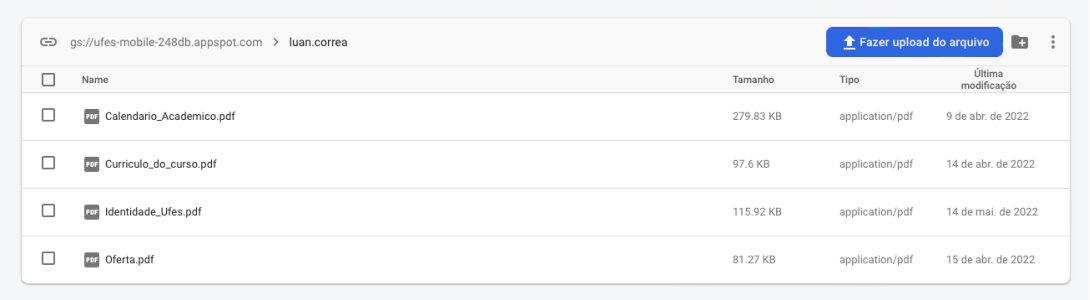

Figura 11 – Gerenciamento de arquivos no cloudstore.

para recuperar os endereços dos documentos. A Listagem 4.19 contém o método da aplicação responsável por resgatar o endereço de um documento, baseado no nome do usuário e do documento solicitado.

Listagem 4.19 – Obtenção de endereços dos documentos no Cloud Storage.

```
export const getDocumentURL = async (userName: string, document: string) => {
    try {
        const storage = getStorage();
        const docName = getDocumentName(document);
        const listRef = ref(storage, `${userName}/${docName}`);
        const url = await getDownloadURL(listRef);
        return url;
    } catch (error) {
        throw new ResponseError(error);
    }
};
```

4.3.5 Notificações

As notificações da aplicação foram implementadas com o uso da biblioteca do próprio Expo, a **expo-notifications**,[10] além de uma funcionalidade do módulo Firestore, citado na Seção 4.3.3. Essa funcionalidade serve para observar mudanças feitas no banco de dados e, quando alguma destas for detectada, será possível resgatar os dados modificados para disparar as notificações com a biblioteca do Expo.

Na documentação oficial da biblioteca disponibilizada pelo Expo, existe um código pronto para configurar as notificações na aplicação. Esta só precisa ser executada no início do aplicativo, e após isso a biblioteca está pronta para poder realizar notificações. A Listagem 4.20 mostra como torna-se simples lançar uma notificação no aplicativo com o método **scheduleNotificationAsync()** da biblioteca, depois de tudo configurado corretamente.

A função que observa dados no banco de dados do Firestore é a **onSnapshot()**. Como parâmetro podem ser passados uma *query* para dizer exatamente qual tabela deve ser observada e uma função que será executada após o recebimento dos dados modificados.

[10] expo-notifications: <https://docs.expo.dev/versions/latest/sdk/notifications/>

Listagem 4.20 – Função para relizar notificações.

```
export async function schedulePushNotification(title: string, message: string) {
    await Notifications.scheduleNotificationAsync({
      content: {
        title,
        body: message
      },
      trigger: { seconds: 1 },
    });
}
```

Figura 12 – Exemplo de notificação na aplicação.

Nesta aplicação, as notificações a serem enviadas aos usuários devem ser disparadas assim que são cadastradas no banco de dados. O conjunto da observação dos dados com o envio das notificações pode ser observado na Listagem 4.21. Também é possível observar um exemplo de notificação na aplicação pode ser visualizado na Figura 12.

Listagem 4.21 – Função para observar dados do Firestore.

```
const unsubscribe = onSnapshot(query, (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    if (change.type === "modified") {
      const data = change.doc.data().messages.slice(-1).shift();
      schedulePushNotification(data.title, data.message);
    }
  });
});
```

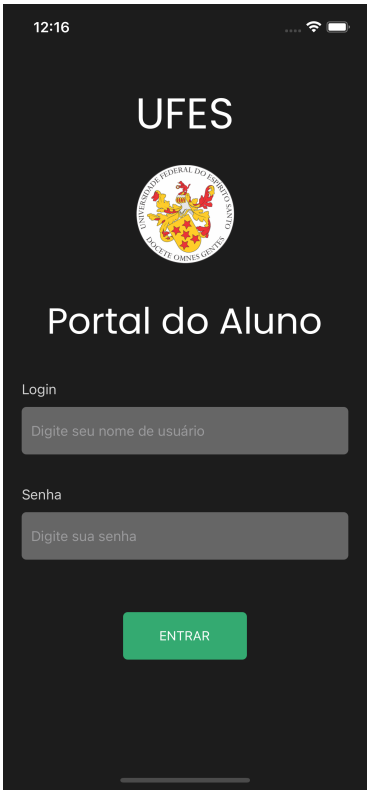

Figura 13 – Tela de Login.

4.4 Apresentação do Sistema

Esta seção possui como objetivo mostrar o resultado final deste projeto, indicando suas telas e funcionalidades desenvolvidas com uma série de capturas de tela. A Figura 13 representa a tela de **Login** da aplicação, que é a tela inicial mostrada ao usuário quando este não possui uma sessão em andamento.

Após fazer Login na aplicação, ou caso o usuário possua uma sessão ativa, este será redirecionado para a navegação por abas, onde a primeira tela é a **Home**. Nesta tela o usuário poderá visualizar as atividades acadêmicas em andamento, bem como o Mural e um progresso geral de disciplinas. Também é possível visualizar um ícone com formato de engrenagem que direciona o usuário para a tela de configurações.

A próxima tela da navegação por abas, é a tela de **Documentos**, representada na Figura 15. Nessa tela é possível selecionar quais documentos se quer acessar. Alguns destes *cards* direcionam o usuário diretamente para a exibição do PDF. Outros passam por algum tela intermediária para permitir filtragem e/ou seleção de informações.

A Figura 16 ilustra um exemplo de tela onde um documento PDF (Calendário Acadêmico) é exibido diretamente. O mesmo ocorre ao acessar o card **Identidade UFES**. Já a Figura 17 ilustra a tela intermediária para acesso aos documentos de oferta de disciplinas. É possível visualizar que existem campos que permitem que seja feita uma filtragem para se acessar o documento desejado. A tela intermediária para seleção também

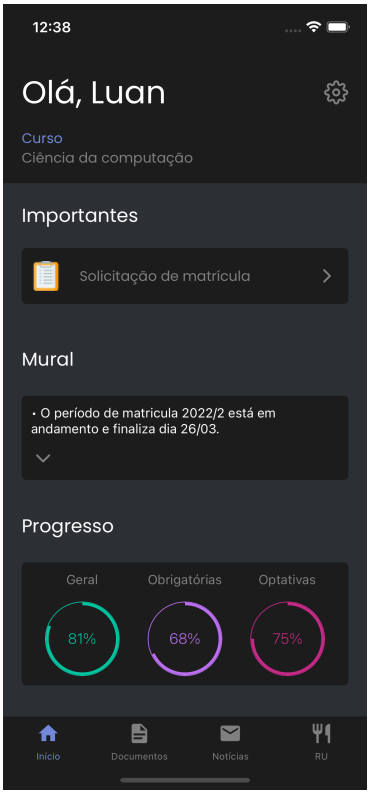

Figura 14 – Tela inicial do usuário logado.

é acessada ao selecionar o card **Relatórios**.

Ainda no aba de Documentos, também existe o *card* **Progresso de disciplinas**, que redireciona o usuário para a tela representada na Figura 18. Nessa tela, o usuário pode conferir, para cada disciplina em atual curso, sua média atual e o número de faltas. Existem colorações diferentes nos números de faltas, para indicar o nível do risco de reprovação.

A próxima aba de navegação é a tela de **Notícias**, representada pela Figura 19. Nesta tela, o usuário poderá visualizar as notícias publicadas pelo colegiado do seu curso. Além disso, caso o usuário deseje, este poderá reagir positivamente à noticia publicada, e será aplicada uma coloração avermelhada no ícone que representa o número de reações.

A última aba possível de ser acessada na aplicação é a tela do **Restaurante Universitário**, representada na Figura 20. Nesta tela é possível visualizar o saldo do RU que o usuário possui, bem como logo abaixo os pratos disponíveis no cardápio do dia. Além disso, também existe um ícone no mesmo *card* do saldo, que direciona o usuário para a tela que permite recarregar o saldo do RU, representada na Figura 21. Nesta tela de recarga, o usuário poderá selecionar o valor que desejar recarregar e a forma de pagamento. Após o usuário clicar no botão de recarregar, será mostrada uma pequena tela contendo o código do pagamento, com a possibilidade do usuário copiar este código. Esta tela está representada na Figura 22.

A Figura 23 representa a tela de configurações da aplicação, onde é possível

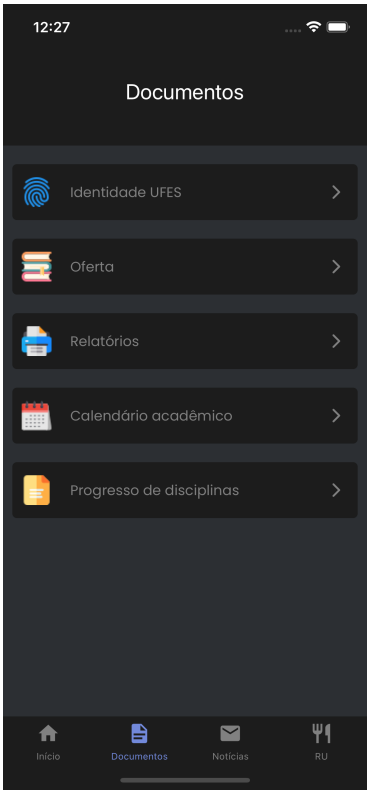

Figura 15 – Tela de documentos.

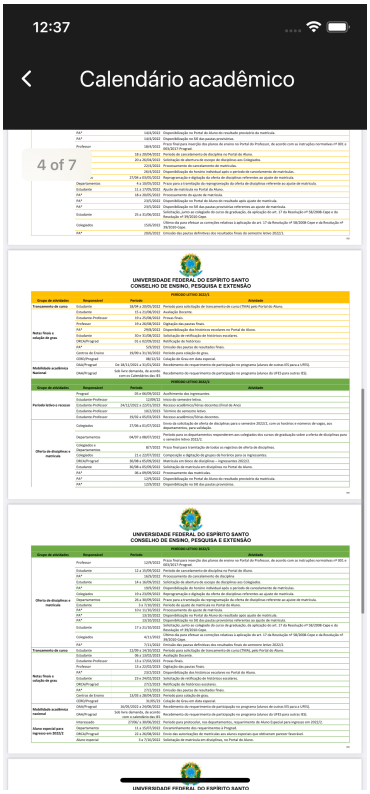

Figura 16 – Exemplo de exibição de documento PDF.

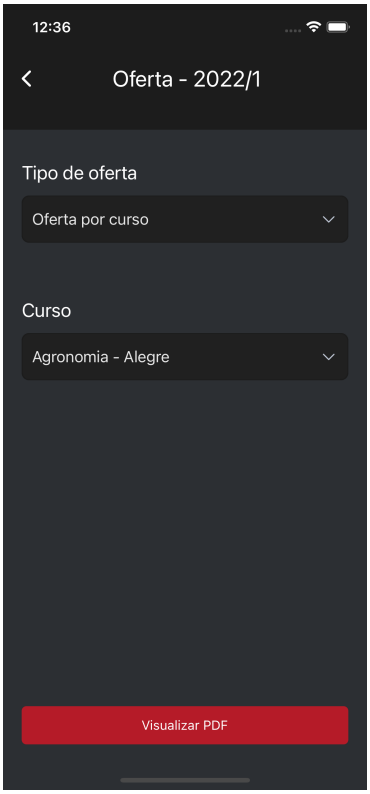

Figura 17 – Tela de seleção de oferta.

Figura 18 – Tela de progresso de disciplinas.

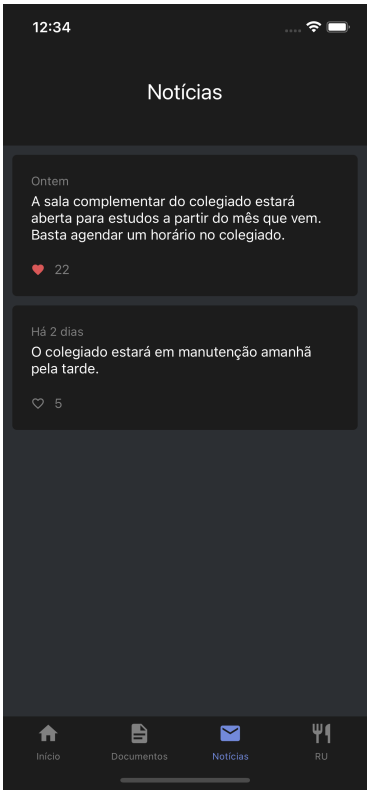

Figura 19 – Tela de notícias.

Figura 20 – Tela do Restaurante Universitário.

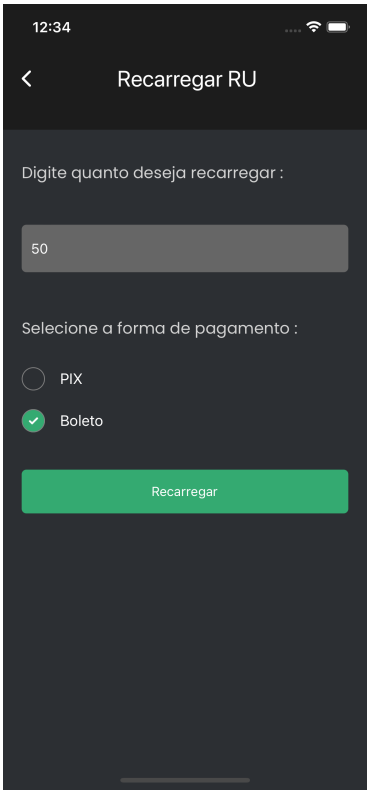

Figura 21 – Tela de recarregamento do saldo do RU.

Figura 22 – Tela com o código de pagamento via boleto.

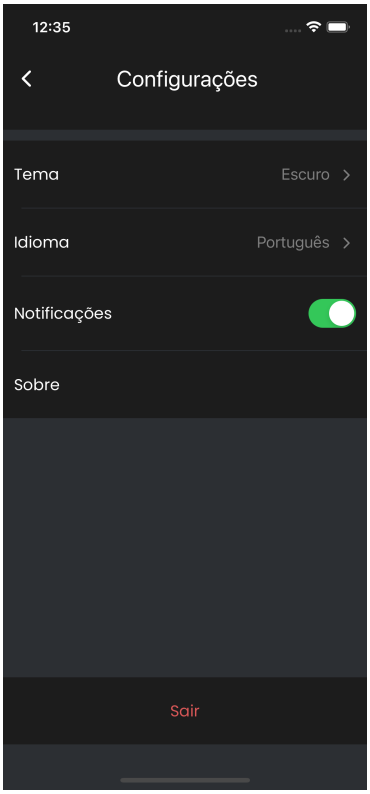

Figura 23 – Tela de configurações.

configurar o tema e idioma da aplicação, bem como ativar/desativar as notificações e realizar Logout. Quando acessadas as configurações tanto do tema, quanto do idioma, o usuário será direcionado para uma tela de seleção, que está representada na Figura 24. Um ícone será mostrado ao lado da opção selecionada para indicar ao usuário, qual a atual configuração.

Na Figura 25 é possível visualizar um exemplo do tema claro aplicado à tela de Login da aplicação. E por fim, na Figura 26 é possível visualizar um exemplo da responsividade que foi aplicada nas telas. A aplicação foi executada em dois dispositivos com dimensões e densidades de pixel bastante diferentes, e ainda assim, foi possível observar uma preservação dos espaçamentos, tamanhos de fonte e de dimensão dos componentes.

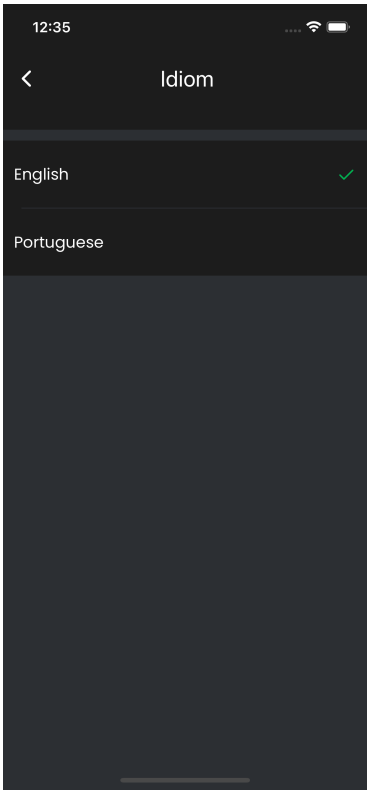

Figura 24 – Tela de seleção do idioma.

Figura 25 – Tela de Login com tema claro.

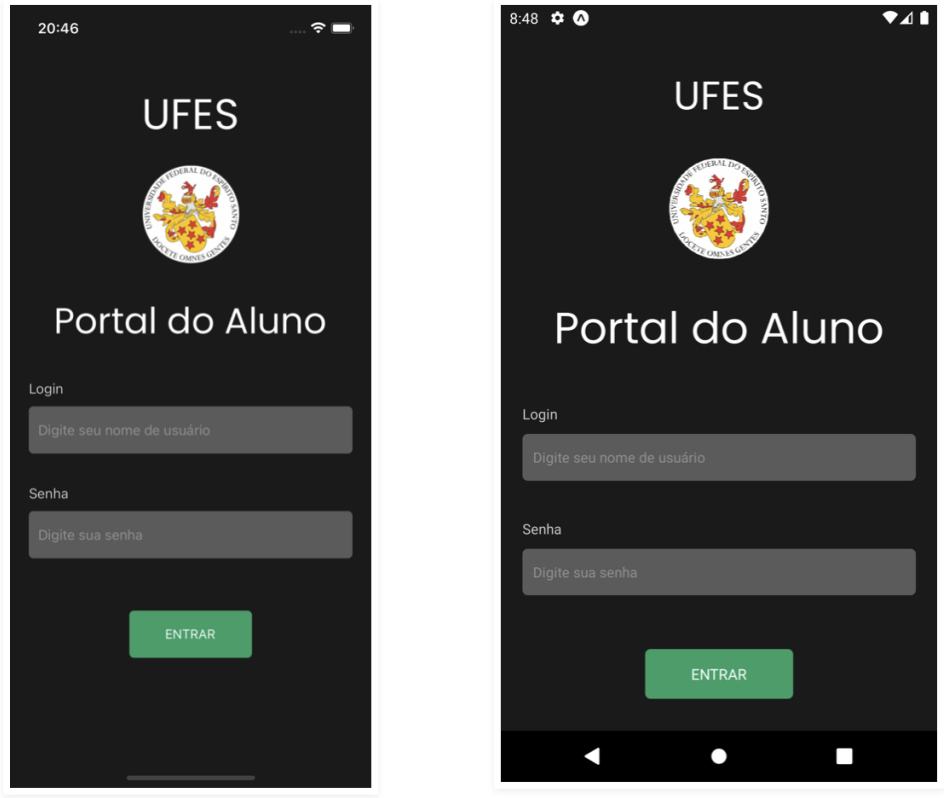

Figura 26 – Aplicação da responsividade em dispositivos diferentes.

5 Considerações Finais

Este capítulo apresenta as conclusões deste projeto, mostrando suas devidas contribuições. Por fim, são apresentadas suas limitações e perspectivas de trabalhos futuros.

5.1 Conclusões

O aplicativo implementando foi imaginado para trazer mais praticidade, agilidade, novas funcionalidades e uma boa experiência de uso no *smartphone*, quando comparado com portal do aluno da UFES na versão Web. Ao mostrar a aplicação para colegas da Universidade, alguns destes até mesmo perguntaram, em tom de brincadeira, quando que o aplicativo estaria disponível para *download*, alegando que seu uso poderia ser benéfico durante o período letivo. Apesar deste projeto ter um cunho imaginário, espera-se que caso um dia a UFES decida de fato criar um aplicativo do seu portal do aluno, para *smartphones*, este projeto possa servir de alguma forma para auxiliar na construção desse.

Quanto aos objetivos propostos no Capítulo 1, estes foram parcialmente concluídos. De fato a implementação da aplicação contemplou todas as funcionalidades que estavam dentro do que foi pensado para esta. Entretanto, o seu desenvolvimento levou mais tempo do que o esperado por motivos de falta de conhecimento prévio das tecnologias utilizadas, complexidade moderada de algumas funcionalidades, e a tentativa de construir a aplicação da maneira mais organizada possível. Isso ocasionou em não ser possível produzir os documentos mencionados nos objetivos, em tempo hábil. Alguns dos artefatos que constariam destes documentos (como diagramas de casos de uso, de classes, de arquitetura), no entanto, estão presentes nesta monografia.

O desenvolvimento deste trabalho foi de extrema importância no quesito experiência. Havia pouco ou nenhum conhecimento nas tecnologias escolhidas para este trabalho, como React Native e Firebase. A experiência e a maior parte dos conhecimentos foi adquirida durante a implementação, onde foram adquiridas diversas noções de desenvolvimento *Mobile*. Noções de design também foram adquiridas com bastante pesquisa e tentativas e erros, pois era parte do objetivo criar uma aplicação visualmente agradável. Além disso, o aprendizado destas tecnologias permitiram que fosse possível ingressar em oportunidades de estágio para desenvolvimento com React Native, e poderá servir para futuras oportunidades.

As principais dificuldades deste trabalho foram a construção da Camada de Controle, com as classes que gerenciam estados globais na aplicação. São muitos detalhes envolvidos, e geralmente as bibliotecas utilizadas para esses fins possuem um moderado nível de complexidade. Portanto, foram dedicadas horas de leitura para se ter um bom

entendimento de como as coisas funcionavam, para finalmente ter condições de implementar a estrutura dessa camada. Outra dificuldade encontrada foi a utilização das funcionalidades disponibilizadas pelo Firebase. Apesar de bem documentadas, estas possuem uma estrutura não muito intuitiva, resultando em várias confusões surgindo na tentativa de utiliza-las. E novamente, foram necessários várias leituras na documentação oficial e em sites alternativos.

Algumas lições bastante significativas foram aprendidas no desenvolvimento deste projeto. Acredito que a principal delas é a importância de saber organizar bem uma aplicação, com uma boa organização de pastas e divisão concisa das camadas. Pois depois de organizada a arquitetura do projeto, se torna muito mais intuitiva a noção de como incluir uma nova funcionalidade, alterar algo já existente e até mesmo aplicar refatorações no código. Neste contexto, outra lição aprendida foi a importância da aplicação de boas práticas de escrita de códigos. Diversas funcionalidades reutilizaram trechos de código, então foi importante escrever códigos com uma mentalidade de torná-los intuitivos, desacoplados e concisos, utilizando princípios de *Clean Code* e SOLID. Por fim, a importância de saber estimar o esforço necessário para desenvolver determinadas funcionalidades também foi uma grande lição deste trabalho. Muitas funcionalidades parecem requerer menos esforço que o necessário, e uma estimativa imprecisa pode ser preocupante em tarefas que possuem prazo para serem entregues, algo muito comum no mercado de trabalho.

5.2 Limitações e Perspectivas futuras

A partir dos resultados alcançados no desenvolvimento desta aplicação, algumas limitações e melhorias puderam ser percebidas, e poderiam ser aplicadas em trabalhos futuros. Dentre as possíveis melhorias e limitações, podem ser listadas:

- As atividades acadêmicas que podem ser visualizadas pelos alunos são apenas botões, mas que não redirecionam o aluno para estas funcionalidades, pois não foram implementadas. A implementação da solicitação de matrícula, por exemplo, possui um moderado nível de complexidade e seria interessante ser implementada em trabalhos futuros;
- A biblioteca utilizada para enviar notificações ao usuário é exclusiva do Expo e isso impossibilita o aplicativo de enviar notificações *push* através do Firebase, por exemplo, que possui um dos serviços mais consolidados no mercado para este tipo de prática;
- O tema claro da aplicação, apesar de possuir um resultado razoável, apenas aplicou inversão de cores no tema original criado, que é o escuro. Porém, o tema claro possui seus próprios padrões de design, que se aplicados, poderiam melhorar a aparência da

aplicação neste tipo de tema;

- A única maneira de se fazer Login na aplicação é digitando manualmente o nome de usuário e a senha. Trabalhos futuros poderiam implementar métodos para permitir que os usuários utilizem impressão digital ou reconhecimento facial para realizar Login, visto que diversos aparelhos possuem nativamente estas funcionalidades;

- A adição de uma seção que integrasse uma relação com os livros que o usuário possui emprestados pela biblioteca, com o tempo restante para devolução, multas, etc, também seria interessante ser implementada em trabalhos futuros;

- A aplicação não possui um controle quanto à mudança de orientação do aparelho nos modos retrato e paisagem. Seria interessante que existisse esse controle para permitir e facilitar a visualização de documentos PDF de larga escala, que são melhor visualizados no modo paisagem.

- A aplicação, por utilizar um banco de dados próprio com dados fictícios de usuários, não tem condições de representar informações reais dos alunos, por não estar conectada com o banco de dados da UFES.

Referências

ABRAMOV, D. React v16.8: O react com hooks. 2019. Disponível em: https://pt-br.reactjs.org/blog/2019/02/06/react-v16.8.0.html. Citado na página 24.

CODERSLANG. What is typescript and why should you use it? 2021. Disponível em: https://learn.coderslang.com/0056-what-is-typescript-and-why-should-you-use-it/. Citado na página 21.

EISENMAN, B. *Learning React Native: Building Native Mobile Apps with JavaScript. O'Reilly Media Inc*, California, 2015. Citado 3 vezes nas páginas 8, 27 e 28.

EL-GAYAR, O. F. et al. *Mobile Applications for Diabetes Self-Management: Status and Potential. Journal of Diabetes Science and Technology*, v. 7, p. 247 – 262, 2013. Citado na página 13.

EXPO. Workflows. 2022. Disponível em: https://docs.expo.dev/introduction/managed-vs-bare/. Citado na página 29.

FALBO, R. A. Engenharia de requisitos: Notas de aula. 2014. Disponível em <http://www.inf.ufes.br/~jssalamon/wp-content/uploads/disciplinas/engsoft/Notas_Aula_Engenharia_Software_Falbo_2014.pdf>. Citado 3 vezes nas páginas 17, 18 e 19.

FLANAGAN, D. Javascript: The definitive guide: Activate your web pages. O'Reilly Media, Inc, 2011. Citado na página 20.

HOLMES, E. et al. *Getting Started with React Native. Packt Publishing*, 2015. Citado 3 vezes nas páginas 23, 27 e 28.

IGHOSEWE, E. Should i use expo for react-native. 2021. Disponível em: https://upstack.co/knowledge/should-i-use-expo-for-react-native. Citado na página 28.

LIMITATIONS. 2022. Disponível em: https://docs.expo.dev/introduction/why-not-expo/. Citado na página 29.

MAHARANA, N. *Cross Platform Mobile Apps and Its Pros and Cons.* 2017. Disponível em <https://medium.com/@netranandamaharana/cross-platform-mobile-apps-and-its-pros-and-cons-9c257ec64e94>. Citado na página 26.

MEHTA, B. M. et al. *Firebase: A Platform for your Web and Mobile Applications. International Journal of Advance Reserach in Science and Engineering*, v. 6, n. 4, 2017. Citado 2 vezes nas páginas 24 e 25.

MORONEY, L. *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform, 1ª edição. Apress*, v. 6, n. 4, 2017. Citado 2 vezes nas páginas 24 e 25.

OCCHINO, T. *React Native: Bringing modern web techniques to mobile.* 2015. Disponível em: <https://engineering.fb.com/2015/03/26/android/react-native-bringing-modern-web-techniques-to-mobile/>. Citado na página 27.

PRESSMAN, R. S. Engenharia de software: Uma abordagem profissional. McGraw-Hill, São Paulo, 2011. Citado na página 19.

ROBBESTAD, S. A. *ReactJS Blueprints. Packt Publishing*, 2016. Citado 2 vezes nas páginas 22 e 23.

SHUBEL, M. What is typescript? 2022. Disponível em: https://thenewstack.io/what-is-typescript/. Citado na página 21.

SILVA, M. M. da et al. Os paradigmas de desenvolvimento de aplicativos para aparelhos celulares. *Revista TIS*, v. 3, n. 2, 2014. Citado na página 25.

SOMMERVILLE, I. Engenharia de software. Prentice Hall, 2011. Citado 2 vezes nas páginas 18 e 19.

VIPUL, A. et al. *ReactJS by Example: Building Modern Web Applications with React: Get up and running with ReactJS by developing five cutting-edge and reponsive projects. Packt Publishing*, 2016. Citado 2 vezes nas páginas 22 e 23.

ZAKAS, N. C. *Understanding ECMAScript 6. The definitive guide for Javascript developers. No Starch Press*, 2016. Disponível em: <http://libgen.rs/book/index.php?md5=bfda74ab195c2f6bb9b563deb4e2a375>. Citado na página 20.