MEC-SETEC

INSTITUTO FEDERAL DE MINAS GERAIS - *Campus* Formiga

Curso de Ciência da Computação

PROTÓTIPO DE UMA *SINGLE PAGE APPLICATION* PARA AUTOATENDIMENTO BIBLIOTECÁRIO USANDO REACTJS

Ana Paula da Silva Cunha

Orientador: Prof. Dr. Bruno Ferreira

Coorientador: Ms. José Pedro Ribeiro Belo

Formiga - MG

2019

ANA PAULA DA SILVA CUNHA

PROTÓTIPO DE UMA *SINGLE PAGE APPLICATION* PARA AUTOATENDIMENTO BIBLIOTECÁRIO USANDO REACTJS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Bruno Ferreira
Coorientador: Ms. José Pedro Ribeiro Belo

Formiga - MG

2019

Cunha, Ana Paula da Silva
004 Protótipo de uma Single Page Application para Autoatendimento Bibliotecário Usando ReactJS / Ana Paula da Silva Cunha. -- Formiga : IFMG, 2019.
74p. : il.

Orientador: Prof. Dr. Bruno Ferreira
Co-orientador: Msc. José Pedro Ribeiro Belo
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1.JavaScript. 2. ReactJS. 3 . MongoDB. 4. RFID. 5. SPA. I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Msc. Naliana Dias Leandro CRB6-1347

ANA PAULA DA SILVA CUNHA

PROTÓTIPO DE UMA SINGLE PAGE APPLICATION PARA AUTOATENDIMENTO BIBLIOTECÁRIO USANDO REACTJS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais-Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 15 de junho de 2019.

BANCA EXAMINADORA

Prof.º Dr. Bruno Ferreira

Prof.º Ms. José Pedro Ribeiro Belo

Prof.º Ms. Fernando Paim Lima

Prof.º Dr. Otávio de Souza Martins Gomes

AGRADECIMENTOS

Em primeiro lugar agradeço à Deus, pela vida e por ter me dado força de chegar até onde cheguei.

Aos meus pais, Sônia e Carlos Roberto, por todo amor, carinho e apoio que sempre deram a vida toda.

Ao meu namorado e coorientador José Pedro, por toda paciência, incentivo, por ter acreditado em mim quando eu mais desacreditava, por não ter me deixado desistir. Além do conhecimento que sempre compartilhou comigo. E principalmente pelo amor e carinho.

Ao meu orientador Bruno, por ter aceitado o convite de me orientar neste trabalho. Por ter cedido um computador, sem isso não teria nem metade do projeto, e por transmitir calma nas horas de desespero.

Á todos os meus amigos e colegas, principalmente a "turma da mesinha", tanto os que ainda estão presentes e os que já se formaram, Ana, Lucas, Vinícius, Elias, Rodrigo, Maria, Arthur, Saulo, Wesley, (...) muito obrigada pelos momentos descontraídos.

Á todos os professores por compartilharem seus conhecimentos.

"O sucesso nasce do querer,
da determinação e persistência
em se chegar a um objetivo.
Mesmo não atingindo o alvo,
quem busca e vence obstáculos,
no mínimo fará coisas admiráveis."
(José de Alencar)

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de uma SPA (*Single Page Application*) para funcionalidades cotidianas de uma biblioteca, tanto administrativas quanto para seus clientes, sendo este último o público alvo principal. Este sistema foi batizado com o nome *Sophia*. O objetivo é que o usuário acesse a aplicação a partir de uma máquina local. O cliente do estabelecimento poderá realizar empréstimos e devoluções de exemplares com seu usuário e senha, já cadastrados no banco de dados, de forma autônoma. Já os administradores, além das funcionalidades dos clientes, também poderão cadastrar exemplares e usuários. A aplicação *Sophia* foi desenvolvida utilizando a linguagem de programação JavaScript, juntamente com a biblioteca do *Facebook*, ReactJS, entre outros módulos necessários. Ela também conta com MongoDB como seu banco de dados e periféricos para auxílio na leitura de etiquetas RFID que estarão fixadas nos exemplares.

Palavras-chave: JavaScript, ReactJS, MongoDB, RFID, SPA.

ABSTRACT

This work presents the development of a prototype of a SPA (Single Page Application) for daily functionalities of a library, both administrative and for its clients, the latter being the main target audience. The developed system was named *Sophia*. The goal is for the user to access the application from a local machine. The customer location may make loans and returns of copies with your username and password, as registered in the database, autonomously. Already administrators, in addition to client features, may also register copies and users. The Sophia application was developed using the JavaScript programming language, along with the *Facebook* library, ReactJS, among other required modules. It also has MongoDB as its database and peripherals for assistance in reading RFID tags that are attached to the copies.

Keywords: JavaScript. ReactJS. MongoDB. RFID. SPA.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BSON	*Binary* JSON
CSU	Caso de Uso
CRUD	*Create*, *Read*, *Update* e *Delete*
DOM	*Document Object Model*
ES6	ECMAScript 6
IoT	*Internet of Things*
JSON	JavaScript *Object Notation*
MVC	Modelo, Visão e Controle
MISO	*Master Input Slave Output*
MOSI	*Master Output Slave Input*
NoSQL	*Not Only SQL*
NOOBS	*New Out Of Box Software*
PU	Processo Unificado
RFID	*Radio-Frequency Identification*
RPi	*Raspberry Pi*
SCLK	Sinal de Clock
SGBD	Sistema de Gestão de Banco de Dados
SPA	*Single Page Application*
SPI	*Serial Peripheral Interface*
SO	Sistema Operacional
SQL	*Structured Query Language*
SSH	*Secure Shell*

UID	*Unique Identifier*
UML	*Unified Modeling Language*
VNC	*Virtual Network Computing*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, livros digitais são usados como uma praticidade, estes podem estar em qualquer dispositivo móvel, podendo ser levado para qualquer lugar, sem a necessidade do usuário sair de casa para realizar o empréstimo do livro. Mas ainda hoje com toda evolução digital tendo todas as suas facilidades, a diversidade do acervo de livros físicos ainda é maior que *e-books*[1].

Bibliotecas atuais, usam algum tipo de sistema digital para a automatização no processo de empréstimo e devolução dos livros, alguns sistemas tendo algumas funções adicionais, tais como reserva de exemplares e renovação de empréstimo, por exemplo. Tais aplicações são operados por bibliotecários do estabelecimento, mesmo com estes sistemas, às vezes o processo de empréstimo/devolução pode ser demorado por ter uma certa quantidade de clientes da biblioteca, esperando para que tal serviço seja realizado.

Computadores têm sido aplicados a uma infinidade de áreas, desde controlar usinas nucleares até desenvolvimento de jogos eletrônicos em telefones celulares. Por causa dessa diversidade de uso, linguagens de programação com objetivos diversos têm sido desenvolvidas (SEBESTA, 2009). Linguagens para desenvolvimento de aplicações específicas como JavaScript, focado para aplicações Web, conhecida como "linguagem do lado cliente" por sua execução se dá diretamente no navegador, independente de servidor (JOEL, 2016).

Este trabalho tem como proposta desenvolver um sistema computacional automatizado de empréstimo de exemplares para bibliotecas, onde o próprio cliente da biblioteca possa realizar esta operação. O sistema, batizado de *Sophia*, é dotado de um leitor de RFID (*Radio-Frequency Identification*) para o reconhecimento do livro. Tal sistema tem como objetivo agilizar o processo de empréstimo e devolução, gerando uma comodidade para o cliente. Deste modo, contribuindo com os usuários e funcionários de bibliotecas.

1.1 Justificativa

A área da tecnologia gerencia recursos ligados a sistemas de automação, tendo como objetivo tornar tarefas cotidianas mais simples e práticas. Esta área busca a melhoria da vida humana, seja através de atividades cotidianas, profissionais ou de entretenimento (SANTOS, 2008).

[1] abreviação do termo inglês *eletronic book* e significa livro em formato digital. Pode ser uma versão eletrônica de um livro que já foi impresso ou lançado apenas em formato digital (SIGNIFICADOS, 2019).

Sistemas de caixa eletrônicos facilitam e agilizam operações bancárias, provendo ao usuário serviços para o pagamento de boletos e convênios, saque de dinheiro, dentre outras operações. Além do mais, é possível configurar estes sistemas para atender clientes fora do horário de expediente do banco, dando maior comodidade aos usuários.

A SPA (*Single Page Application*) para autoatendimento bibliotecário de empréstimo pode ser comparado a um caixa eletrônico. Com este, o usuário pode fazer o empréstimo sozinho, assim diminuindo a sobrecarga de tarefas do bibliotecário e oferecer comodidade ao cliente.

1.2 Objetivo

O projeto apresentado neste documento, tem como objetivo criar um protótipo de um sistema bibliotecário para facilitar e agilizar a realização de empréstimo em uma biblioteca, deste modo reduzindo a sobrecarga do funcionários. Tendo como objetivos específicos:

- A implementação de uma aplicação de uma só pagina (*Single Page Application*, SPA);
- Interface intuitiva para interação com os usuários;
- Agilizar e facilitar o processo de empréstimo;
- Trazer comodidade aos usuários;
- Minimizar a sobrecarga dos funcionários de bibliotecas; e
- Prover periféricos (*hardware*) com a finalidade da leitura de códigos.

1.3 Estrutura do trabalho

Este trabalho é composto por seis capítulos, sendo que este é o primeiro, onde fez-se as apresentações das ideias iniciais do projeto, com os objetivos e a justificativa.

No Capítulo 2, é apresentado a fundamentação teórica sobre sistemas Web voltado para bibliotecas, linguagem JavaScript e um pouco sobre a biblioteca ReactJS, uma visão geral sobre banco de dados relacionais e banco de dados não relacional MongoDB, Interface de Programação de Aplicações, para auxílio entre servidor e aplicação, dispositivo para leitura de *tags* RFID.

No Capítulo 3 os materiais e a metodologia utilizados para desenvolver a aplicação são abordados. No Capítulo 4 é exposto a modelagem do sistema, modelagem e manipulação

do banco de dados, além da leitura do RFID pelo sistema. Já o Capítulo 5 contém os resultados da montagem do circuito usado para leitura da *tag* e as interfaces gráficas.

Por fim, no Capítulo 6 são apresentadas as considerações finais e discussões sobre este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Alguns conceitos são necessários para entendimento da aplicação que será apresentada nesta monografia. Como é uma aplicação Web para biblioteca voltada para seus clientes, primeiro é preciso definir o conceito de *frontend*, a parte que o usuário irá interagir, *backend*, onde tratará todas funcionalidades do sistema e banco de dados, onde os dados serão armazenados.

2.1 Dia

Dia é um programa de código aberto criado por Alexander Larsson. Foi inspirado no programa comercial do Windows "Visio". Pode ser usado para criação de diferentes tipos de diagramas, como UML, relacionamento de entidade, fluxogramas, diagramas de redes entre outros. Suporta formatos EPS, SVG, XFIG, WMF e PNG, sendo possível também adicionar suporte para novas formas escrevendo em arquivos XML simples, e usando SVG para desenhas dormas (WIKI, 2018).

2.2 Sublime Text

Sublime é um editor de texto sofisticado para código, marcação e prosa. É construído a partir de componentes personalizados, proporcionando uma capacidade de resposta incomparável. Tem um ecossistema de pacotes poderoso, que armazena funcionalidades usadas com pouca frequência, como classificar alterar a sintaxe e alterar configurações de recuo. Uma plataforma cruzada[1] disponível para Mac, Windows e Linux (SUBLIME, 2019).

2.3 Aplicações Web

Aplicações Web são trabalhos multidisciplinares, pois não envolve apenas a programação em si, como também a modelagem da base que armazenará os dados do software, o planejamento antecipado da funcionalidades que deve ter neste sistema, entres outras preocupações que podem aparecer no andamento do desenvolvimento da aplicação.

Na subseção 2.3.1 será apresentado alguns sistemas bibliotecários, estes tendo a característica nacional, como o software livre Gnuteca, quanto internacional, como

[1] Plataforma que alcança seus consumidores através de qualquer dispositivo/sistema operacional que eles utilizam (SILVA, 2013)

MicroISIS que é popular no Brasil e no mundo. Demais subseções apresentam algumas das tecnologias utilizadas para desenvolvimento do software apresentado neste trabalho.

2.3.1 Sistemas Web para Bibliotecas

Juntar registros em um único lugar, é considerado a melhor forma para facilitar a ordenação e acesso à registos, formando-se coleções. As coleções, que foram denominadas "bibliotecas", que como definição é a existência de alguma forma de organização para encontrar o que é desejado (MILANESI, 2002).

Softwares voltados para bibliotecas surgiram graças aos recursos de Tecnologia da Informação existentes nas Instituições de ensino e empresas (DAMASIO; RIBEIRO, 2006).

2.3.1.1 MicroISIS

O CDS/ISIS é um sistema genérico para gerenciamento de bases de dados não-numéricas, foi desenvolvido, para computadores de grande porte (MIKI, 1989). O MicroISIS é a derivação do CDS/ISIS, para microcomputadores, desenvolvido pela UNESCO para tratamento e gerenciamento de informações (INFOISIS, 2004).

Segundo Damasio e Ribeiro (2006), MicroISIS é um dos softwares mais popularizados para bibliotecas, no Brasil e no mundo, devido à não necessidade de pagar licenças. Sua principal função é organizar dados catalogados e distribuí-los em bases de dados.

2.3.1.2 GNUTECA

Outro sistema para bibliotecas é o GNUTECA, um software brasileiro criado pela Solis[2] LTDA, que está no ar desde 2001 (SOLIS, 2018).

> O GNUTECA é um sistema para automação de todos os processos de uma biblioteca, independentemente do tamanho de seu acervo ou da quantidade de usuários. O sistema foi criado de acordo com critérios definidos validados por um grupo de bibliotecários e foi desenvolvido a partir de testes nesta biblioteca real, a do Centro Universitário Univates (DAMASIO; RIBEIRO, 2006).

O software brasileiro é livre e também respeita o MARC21, que é um formato de catalogação bibliográfica (SOLIS, 2018). Outra característica, é sua base no ambiente CDS/ISIS, com isso tento uma fácil migração de acervos deste tipo (DIAS; MIRANDA, 2013)

[2] Solis/Gnuteca <https://www.solis.com.br/gnuteca>

2.3.1.3 OpenBiblio

O OpenBiblio[3] é um Sistema de Automação de Biblioteca (SAB) de fácil uso, que foi escrito em PHP (OPENBIBLIO, 2017). Seu objetivo é gerenciar e informatizar bibliotecas, de modo que facilite as atividades dos bibliotecários. O sistema também faz o uso do banco de dados MySQL, foi desenvolvido em plataforma Linux mas é um sistema flexível quanto à outros sistemas operacionais (DIAS; SILVA, 2010).

2.3.2 Node.js e NPM

O Node.js surgiu graças às necessidades que ocorreram ao desenvolver aplicações Web (RUBENS, 2017). É uma plataforma que permite criar o próprio servidor e construir aplicações em cima dele (HOLMES, 2016). Pode ser usado para qualquer fim e instalado em qualquer máquina, assim permitindo ser executado em computadores pessoais, microcontroladores e até mesmo *smartphones* (POWERS, 2017).

Uma das vantagens de programar usando o Node.js, é que toda a aplicação, *backend* e *frontend*, será escrita em apenas uma linguagem, o JavaScript (HOLMES, 2016). Como o JavaScript funciona com uma única *thread*, o Node emula um ambiente assíncrono, mas aceita a maioria das funções para manipulação de sistemas de arquivos (POWERS, 2017).

Para começar a usar o Node, é necessário tê-lo instalado na máquina. Para isso é preciso entrar no site do Node.js[4] e seguir as instruções de instalação como é recomendado de acordo com a plataforma escolhida (POWERS, 2017).

Node Package Manager (NPM), é o gerenciador de pacotes do Node.js, que é instalado junto com ele (HOLMES, 2016). Com o NPM a instalação de módulos de terceiros é mais fácil (POWERS, 2017), basta apenas usar `npm install <nome-do-pacote>` para instalar o pacote desejado.

2.3.2.1 Nodemon

Módulo Nodemon funciona com um *watcher*, ele fica observando toda vez que um arquivo do projeto é alterado e reinicia o processo do Node, evitando de recarregar o servidor sempre que é feita uma alteração (RUBENS, 2017).

2.3.3 JavaScript

JavaScript é uma linguagem de *scripting* orientada à objetos (FLANAGAN, 2004), com código embarcado em documentos HTML e interpretado pelo navegador, voltada para programação Web. O uso de seu interpretador é mais comum em navegadores, apesar de

[3] OpenBiblio <http://obiblio.sourceforge.net/>
[4] Site Node.js <https://nodejs.org/en/>

poder ser embarcado em muitas aplicações (SEBESTA, 2009). Também conhecido como "Mocha", "LiveScript", "JScript"e EMCAScript", é uma das linguagens mais populares do mundo, por ter o papel de "linguagem de script pra o ambiente Web"(CROCKFORD, 2001).

É uma linguagem de alto nível, interpretada e dinâmica, pode ser usada de modo geral que à torna robusta e eficiente (FLANAGAN, 2004). Segundo Crockford (2001), comparado com o C, o JavaScript consegue um expressivo poder de performance e dinamismo.

Ao desenvolver um aplicativo Web, é necessário levar em conta como o usuário irá interagir com ele e qual será a melhor maneira desta interação acontecer. Para isso existem as bibliotecas de estrutura ou *framework*, que permitem fazer mais com menos código. Elas constroem uma nova API de nível mais alto para programação no lado do cliente, sobre as APIs padrão e oferecidas pelos navegadores Web (FLANAGAN, 2004).

2.3.4 ReactJS

ReactJS é uma biblioteca JavaScript, do Facebook, que evoluiu originalmente de *framework* chamado BoltJS e FaxJS (KOPPALA, 2018). A biblioteca tem seu código aberto, voltada para aplicações web e sites, que tem como objetivo superar desafios encontrados no desenvolvimento de aplicativos de uma só página (SPA), com códigos flexíveis e eficientes (VIPUL; SONPATKI, 2016).

ReactJS é orientado à componentes. Estes podem ser encapsulados, e também gerenciam o próprio estado (REACT, 2019). Não é um *framework*, mas pode ser combinado com alguns, como AngularJS, Ember e Meteor (ROBBESTAD, 2016).

Segundo Robbestad (2016), ReactJS pode ser definido como o *V* do MVC (Modelo, Visão e Controle). Marcas conhecidas como *Instagram*, *Yahoo*, *Netflix* e *AirBnB* usam React, seja para Web (ReactJS) e/ou na versão de aplicativo para *smartphones* (React Native) (VIPUL; SONPATKI, 2016). A Figura 1, mostra estatísticas retiradas do site W^{3}Techs[5], que fornece informações sobre vários tipos de tecnologias na Web (W3TECHS, 2019), sobre três das bibliotecas JavaScript, Angular, Vue.js e inclusive React.

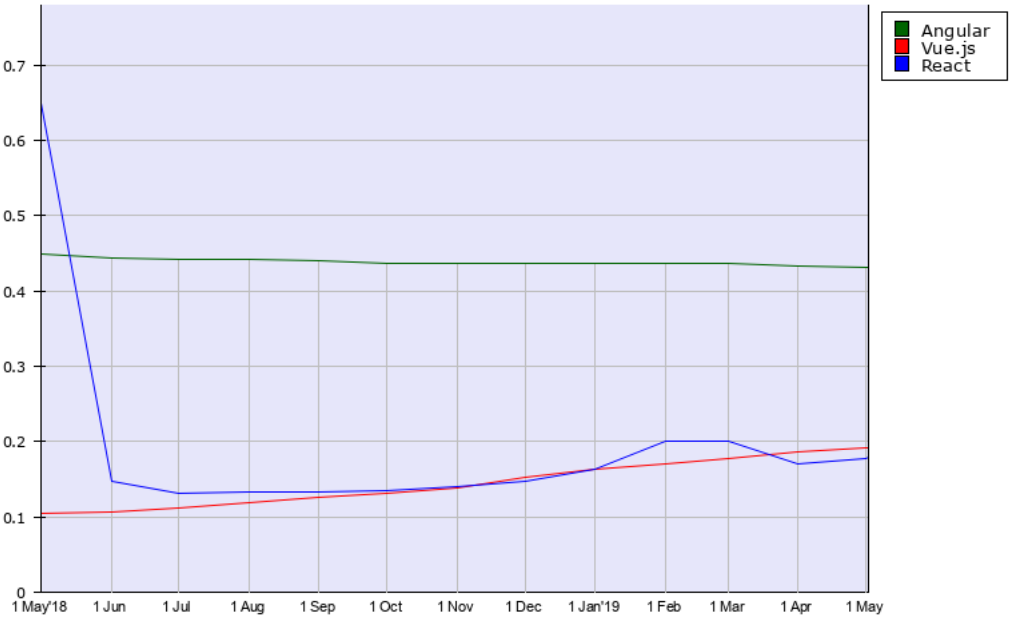
Segundo a Organização React (2019), a biblioteca suporta os navegadores populares, excluindo os mais antigos que não oferecem suporte aos métodos ES5.

2.3.4.1 React DOM

Para introduzir o que é realmente o React DOM, é preciso saber o que significa 'DOM'. O autor Wood et al. (1998), descreve que o Modelo de Objeto de Documento, do

[5] W^{3}Techs <https://w3techs.com/>

Figura 1 – Diagrama com tendência histórica na quantidade de sites que usam as tecnologias selecionadas, Angular, Vue.js e React.



Fonte – W3Techs (2019).

inglês *Document Object Model* (DOM) é um fornecedor de uma mescla padrão de objetos para representar documentos HTML e XML.

Para desenvolver uma aplicação React, não é apenas necessário usar a biblioteca 'react', é preciso usar o React DOM. Esta biblioteca é usada para renderizar o interface de usuário no *browser* (BANKS; PORCELLO, 2017), além de fornecercont métodos específicos do DOM que podem ser usados no nível superior para, se necessário, sair do modelo do React (REACT, 2019).

2.3.4.2 Material-UI

O *framework* de interface de usuário React mais famoso do mundo, Material-UI, um projeto da MIT de código aberto licenciado, onde os componentes implementam o Material Design do Google (MATERIAL-UI, 2019). Os autores do *framework* apontam que os componentes do Material-UI funcionam de forma isolada, que são autossustentáveis e não precisam de nenhuma *style-sheet* global.

2.3.4.2.1 Color Tool

Color Tool é uma ferramenta desenvolvida pela equipe Material Design do Google, que permite criar uma paleta de cores tema de uma aplicação, com qualquer combinação de cor, permitindo também compartilha-la (GOOGLE, 2019).

2.3.4.3 Styled Components

Permite escrever o código CSS para estilizar os componentes, remove o mapeamento entre componentes e estilos, e por ser compatível com React e React Native (para dispositivos móveis), é o ideal para aplicativos universais (COMPONTENTS, 2016).

2.3.4.4 React Router

Em uma SPA, tudo acontece na mesma página e o processo de roteamento é o que define os *endpoints* para as requisições dos seus clientes. Como o React não vem com um sistema de roteamento padrão, a solução deste problema foi resolvida por dois engenheiros que criaram o React Router, que foi adotado pela comunidade ReactJS (BANKS; PORCELLO, 2017).

O React Router é um conjunto de componentes de navegação que integram uma aplicação de forma declarativa (TRAINING, 2019). Facilita a navegação por links sem recarregar a página, por meio de componentes (SOUTO, 2018).

2.3.4.5 Redux

É um contêiner de estado previsível para aplicativos JavaScript (REDUX, 2019). Gerencia componentes de forma eficaz, ajudando a conectar os componentes React com o Servidor (RIBEIRO, 2019), deste modo repassando toda a informação para que o servidor armazene no banco de dados, ou requisitando dados.

Centraliza o *state* e a lógica da aplicação, o primeiro comporta como variável, onde armazena toda alteração em um campo de texto, botão, ou qualquer outro componente. Trazendo recursos como desfazer/refazer (REDUX, 2019), facilitando a utilização do *state* por outros componentes (RIBEIRO, 2019).

2.3.5 Express

É um dos módulos mais famosos para desenvolvimento de servidores Web (RUBENS, 2017). Express é um *framework* para aplicações Web, e também um módulo do Node.js (HOLMES, 2016). Por se encaixar muito bem ao Node, torna a programação para aplicações Web fácil (HAHN, 2016). É um projeto da Fundação Node.js que fornece uma camada de recursos fundamentais para aplicações Web, sem ofuscar os recursos do Node.js (EXPRESSJS,).

Holmes (2016) diz que, uma de suas vantagens é não determinar a maneria que deve ser usado. Uma das possibilidades de uso, é para tratar rotas de páginas, de modo simples, e de acordo com as necessidades da aplicação, pode-se interligar vários módulos à ele (RUBENS, 2017).

2.3.5.1 CORS

CORS (*Cross-Origin Resource Sharing*) é uma pacote Node.js que fornece um *middleware* Connect/Express (EXPRESSJS, 2019). Permite que um site acesse recursos de outro site mesmo estando em domínios diferentes (JACQUES, 2016).

2.3.5.2 Mongoose

É a modelagem de objetos do MongoDB para Node.js (MONGOOGSEJS, 2011). O MongoDB, não "conversa" diretamente com o *frontend*, para comunicação é usado apenas o Mongoose, que repassa para Node/Express (HOLMES, 2016).

> O Mongoose fornece uma solução direta e baseada em esquema para modelar os dados do seu aplicativo. Ele inclui conversão de tipo incorporada, validação, criação de consulta, ganchos de lógica de negócios e muito mais (MONGOOGSEJS, 2011).

2.3.5.3 Body Parser

É um *middleware* responsável por transformar a requisição do cliente em um objeto JSON, tratando-o e também os seus dados (CODE, 2009), com o objetivo de garantir que os dados a serem enviados sejam JSON (BERA; MINE; LOPES, 2015).

2.3.5.4 Axios

Suportado por vários navegadores, o Axios é um cliente HTTP para Node.js, baseado na API Promise (AXIOS, 2019). Manda requisições REST para o servidor da aplicação e renderiza o que foi recebido (RIBEIRO, 2019).

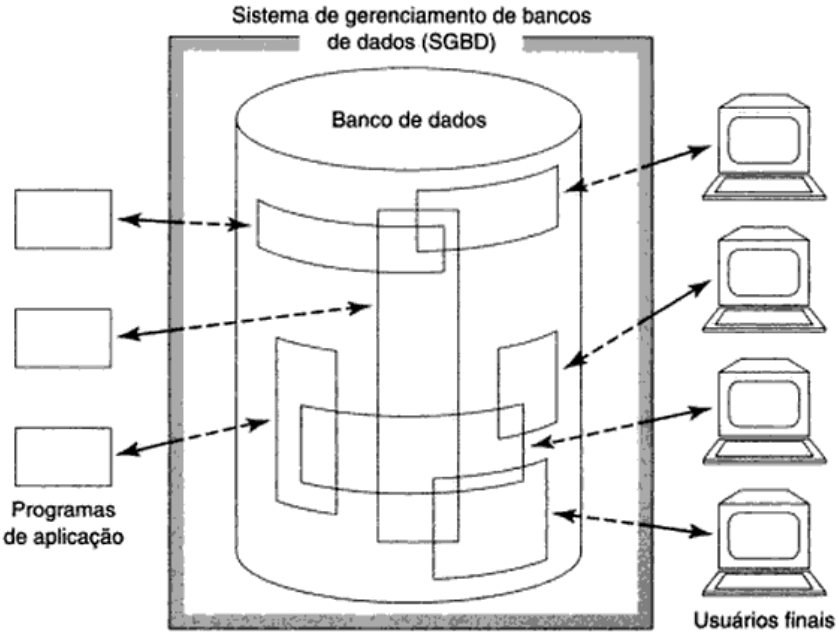
2.4 Banco de Dados

Um banco de dados é aquele que guardará toda e qualquer informação que seja necessária ser acessada em dado momento, seja como consulta ou para modificação/atualização dos dados (DATE, 2004). Um exemplo simples a ser aplicado, é um estabelecimento comercial, onde dados genéricos em caso de um produto seriam: código, nome, preço e quantidade. O código seria para identificar qual produto especificamente. O nome seria nome de fato do produto para descrever o que é. O preço para determinar um valor e quantidade para auxiliar no controle de estoque. Neste caso seria apenas dados genéricos, mas ao especificar qual o tipo de local esse banco será aplicado, é necessário acrescentar algumas informações para o produto, como por exemplo validade, em caso de alimentos, ou até mesmo informações sobre funcionários, vendas, entre outras, além do produto.

Para fazer todo o gerenciamento de um banco de dados, é necessário ter um sistema, para isso exsitem os Sistemas de Gestão de Banco de Dados (SGBDs). Segundo Date

(2004), estes sistemas envolvem quatro componentes principais: dados, hardware, software e usuários, como pode ser visto na Figura 2.

Figura 2 – Representação simplificada de um sistema de banco de dados.



Fonte – Date (2004).

Para a manipulação de um banco de dados é necessário usar uma linguagem. Uma das linguagens mais conhecidas para SGBDs é a SQL (*Structured Query Language*), que por muitos produtos a aceitarem ela se tornou a linguagem padrão de banco de dados relacional (SILBERSCHATZ; SUNDARSHAN; KORTH, 2016). Bancos de dados relacionais são aqueles que usam tabelas para diferenciar seus componentes, como no exemplo usado anteriormente, produto, funcionário e venda seriam três das tabelas existentes. Como são chamados de *relacionais*, originando-se de termo matemático, é a relação que existe entre as tabelas. Quando acontece esta relação, é adicionado uma *chave* para simbolizá-la, como se fosse um ponteiro.

Além dos bancos de dados relacionais, existem os bancos de dados não relacionais. Em sistemas relacionais, todos os dados são organizados em tabelas, já em bancos não relacionais, estes podem ser organizados em outras estruturas que não sejam apenas tabelas. Além desta característica, estes sistemas por não serem relacionais, não exigem a necessidade de tererem uma referência (DATE, 2004).

2.4.1 MongoDB

O MongoDB é um banco de dados NoSQL (*Not Only SQL*) orientado a documentos, conhecido por ser rápido e escalonável (HOLMES, 2016). Por motivos de falta de

padronização do JSON, o MongoDB trabalha com BSON, que é uma extensão binária do JSON. BSON suporta alguns formatos além dos já descritos do JSON (BOAGLIO, 2017):

- MinKey, MaxKey, Timestamp - utilizado pelo MongoDB;
- BinData - array de bytes para dados binários;
- ObjectId - identificadores único de um registro do MongoDB;
- Date - data, Expressões regulares

Figura 3 – Ranking de SGBDs mais usados em Maio de 2019.

Rank			DBMS	Database Model	Score		
May 2019	Apr 2019	May 2018			May 2019	Apr 2019	May 2018
1.	1.	1.	Oracle	Relational, Multi-model	1285.55	+5.61	-4.87
2.	2.	2.	MySQL	Relational, Multi-model	1218.96	+3.82	-4.38
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1072.19	+12.23	-13.66
4.	4.	4.	PostgreSQL	Relational, Multi-model	478.89	+0.17	+77.99
5.	5.	5.	MongoDB	Document	408.07	+6.10	+65.96
6.	6.	6.	IBM Db2	Relational, Multi-model	174.44	-1.61	-11.17
7.	↑ 8.	↑ 9.	Elasticsearch	Search engine, Multi-model	148.62	+2.62	+18.18
8.	↓ 7.	↓ 7.	Redis	Key-value, Multi-model	148.40	+2.03	+13.06
9.	9.	↓ 8.	Microsoft Access	Relational	143.78	-0.87	+10.67
10.	↑ 11.	10.	Cassandra	Wide column	125.72	+2.11	+7.89

Fonte – DB-Engines (2019).

MongoDB vem ganhando mais espaço no mercado, como pode ser visto na Figura 3, que se trata do ranking de SGBDs mais populares, divulgado pelo site DB-Engines, que é atualizado mensalmente (DB-ENGINES, 2019). Empresas grandes como Bosh e Expedia usam o MongoDB, pelo sua rapidez, baixo custo e facilidade com que o banco cresce de acordo com a necessidade (MONGODB, 2019).

2.4.1.1 JSON

JavaScript *Object Notation* (JSON) é uma formatação leve, baseado em texto e também é uma linguagem independente de troca de formato de dados, derivada do padrão ECMAScript (CROCKFORD, 2006). Foi criado por Douglas Crockford como uma solução para o problema de falta de simplicidade para enviar informações de um servidor para um *browser* (BOAGLIO, 2017).

JSON pode representar quatro tipos primitivos (*null*, *Boolean*, *Number* e *String*) e dois tipos estruturados (*Array* - lista ordenada - e *Object* - um *array* não ordenado com itens do tipo chave-valor, que são strings distintas no mesmo objeto) (CROCKFORD, 2006).

2.5 API

Com a chegada de aplicações disponibilizadas apenas na Web, a necessidade de comunicação entre aplicações surgiu, resultando em Interface de Programação de Aplicações, API (*Application Programming Interface*). API é um conjunto de rotinas e padrões documentados por uma aplicação, para que outras consigam usar suas funcionalidades (PIRES, 2017). Deste modo tornando possível que aplicações se comuniquem (HOLMES, 2016).

2.5.1 REST

HTTP é o principal protocolo de comunicação Web. Nos anos 2000, um dos principais autores do protocolo sugeriu o uso de novos métodos HTTP que visam resolver o problema de requisições do protocolo (PIRES, 2017).

Requisição HTTP do tipo Transferência de Estado Representacional, do inglês *REpresentationl State Transfer* (REST), não é orientado a estado, ou seja, uma API do tipo REST é uma interface de comunicação com aplicação que não guarda estado da sessão/usuário, usada para criar uma interface com o banco de dados (HOLMES, 2016).

Holmes (2016) ainda acrescenta que solicitações HTTP são enviadas usando os métodos **POST** (cria novos dados), **GET** (Ler dados), **PUT** (atualiza dados) e **DELETE** (remove dados). Estes métodos que também correspondem as funções CRUD (*Create*, *Read*, *Update* e *Delete*) de banco de dados. Um sistema que tem princípios REST é chamado RESTfull (PIRES, 2017).

2.5.2 Postman

É um ambiente de desenvolvimento de API, muito usada para testes manipulação de banco de dados, que foi criada como um projeto paralelo e cresceu rapidamente, tornando um dos aplicativos mais populares do Google Chrome[6], permitindo que equipes colaborem em tempo real e que mantenham o projeto organizado(POSTMAN, 2019).

2.6 Arquitetura e Módulo

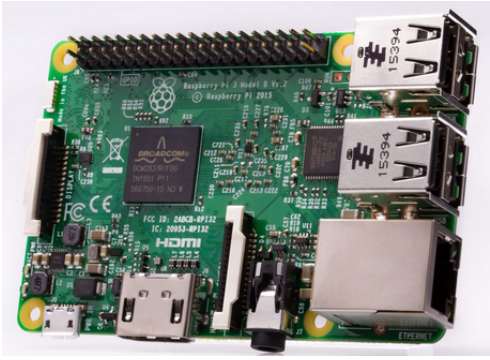
Arduino é um microcontrolador, uma pequena placa com arquitetura de Harvard, lançada em 2005 para ajudar estudantes de design, a ferramenta é voltada para desenvolvimento de produtos IoT, usado para inovar desde brinquedos até mesmo em veículos autônomos entre outros (ARDUINO, 2019). Apesar de ser a ferramenta de prototipagem eletrônica mais popular, não há um sistema operacional, ao contrário do Raspberry Pi.

[6] Postman para Google Chrome - <https://bit.ly/2htrWv2>

2.6.1 Raspberry PI

O nome *Raspberry*, vem da tradição de microcomputadores clássicos como *Tangerine*, *Apricot*, *Acorn* e também as marcas BlackBerry e Apple que também levam nomes de frutas, para um maior reconhecimento, já a o segundo nome, *PI*, é por conta de sua programação ser na linguagem *Python* (UPTON; HALFACREE, 2014).

Figura 4 – Raspberry Pi 3 Model B.



Fonte – RaspberryPi (2019)

O Raspberry PI é um mini computador, de arquitetura Von Neumann, não é um dispositivo de consumo e, dependendo da aplicação a se fazer com o pequeno computador, será necessário tomar várias decisões a respeito dos periféricos e softwares a serem instalados e configurados nele (RICHARDSON; WALLACE, 2013).

Carrega um sistema operacional Linux, o *Raspbian* (oficialmente recomendado), e atualmente é muito usado em IoT (*Internet of Things*) para automação residencial. Apesar de ser pequeno, é uma ferramenta poderosa para programadores e apreciadores da Eletrônica Digital (VUJOVIĆ; MAKSIMOVIĆ, 2015).

2.6.2 RFID

Identificação por rádio frequência, do inglês *Radio Frequency Idendification* (RFID), é uma tecnologia de curto alcance usada para comunicar informações digitais entre uma localização fixa e um ou mais objetos móveis (LANDT, 2005). O autor Finkenzeller (2010), diz que os sistemas RFID estão intimamente relacionados com cartões inteligentes, assim como eles, no sistema de identificação por rádio frequência, os dados são armazenados em um dispositivo eletrônico de transmissão de dados.

Por existir vários tipos de RFID, pode-se dividir em classe passiva e ativa. Algumas tags necessitam de usar uma fonte de energia, que podemos chama-las de tags ativas (Figura 5). Um exemplo de tag ativa é um LoJack[7], acoplado em um carro, com ajuda de um GPS e um celular, pode localizar o carro furtado (WANT, 2006).

[7] Sistema para recuperar carros roubados <https://www.lojack.com/>

Figura 5 – Tags RFID ativas e passivas.

Fonte: Retirado do site Raviraj Technologies[8]

Want (2006) ainda completa que, as tags passivas tem uma vantagem, por não precisarem de algum tipo de fonte, tornando seu uso barato, pois não precisam de manutenções. Etiquetas deste tipo, há três partes: uma antena, um chip semicondutor que se conecta com a antena e encapsulamento.

2.7 Módulo Leitor RFID (MFRC522)

A leitura de tags RFID é feita por um módulo, e neste caso o MFRC522. Segundo RoboCore (2018), este é o chip contido no leitor, que foi criado pela NXP *Semiconductos*, recebe o nome de MIFARE, com padrão ISO/IEC 14443 tipo A de 13,56 MHz. Esta frequência deixa a característica nas *tags* de passivas e ativas, isto é, lê a ID do cartão e também pode armazenar dados (ROBOCORE, 2018).

[8] Raviraj Technologies <https://www.ravirajtech.com/rfid-tags.html>

3 MATERIAIS E MÉTODOS

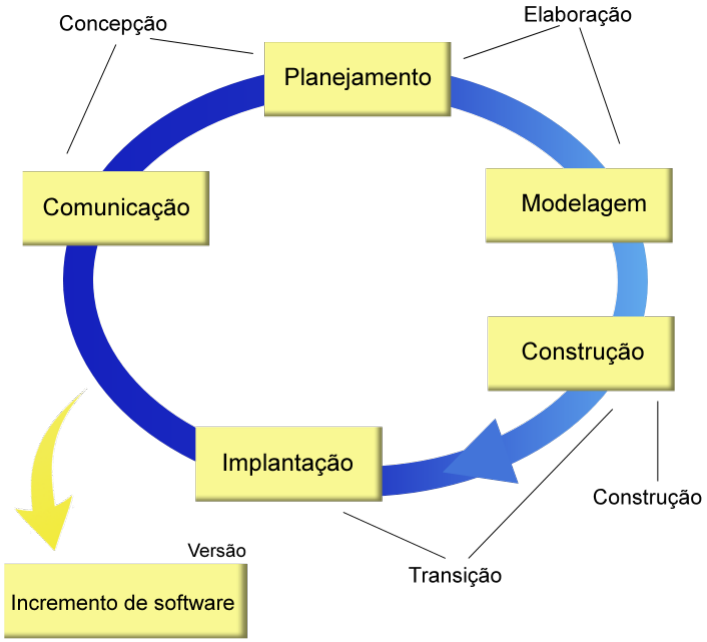
Desenvolver um software vai além dos costumes de um programador, é preciso enxergá-lo como um todo, quais usuários irão utilizá-lo, o quão grande será os dados a serem armazenados, entre outros fatores (RUBENS, 2017). Neste capítulo serão apresentados os materiais e métodos utilizados neste trabalho.

3.1 Metodologia

A metodologia seguida neste trabalho foi inspirada no processo unificado (PU). Este tipo de processo é uma estrutura genérica, podendo ser customizado de acordo com as necessidades específicas do projeto (OBJETOS; JUNIOR, 2004). Utiliza a Linguagem de Modelagem Unificada (Unified Modeling Language, UML) para preparar os artefatos do sistema (CALÇADO, 2007). O ciclo de vida do sistema constitui de várias interações, formado por quatro fases: Concepção, Elaboração, Construção e Transição. Cada fase sendo subdividida em fluxos de requisitos, análise, projeto, implementação e teste (OBJETOS; JUNIOR, 2004). A Figura 6 mostra como é o ciclo de vida de um PU.

No processo de Concepção se define o escopo do sistema, esboço da arquitetura, estima-se os custos e cronograma. A Elaboração captura-se os requisitos funcionais da

Figura 6 – Fases do Processo Unificado



Fonte – Makino (2009)

aplicação que não foram capturados da fase anterior, faz-se a expansão do esboço da arquitetura para uma arquitetura de desenvolvimento do sistema. A fase de Construção é onde o produto é construído. Já a Transição é onde o produto tem a primeira versão entregue ao cliente para teste (OBJETOS; JUNIOR, 2004).

3.2 Métodos Utilizados

A partir das necessidades que foram surgindo ao desenvolver uma aplicação em JavaScript usando Node.js, a medida que o programa foi ganhando vida, foi necessário optar pelo uso/instalação de algumas dependências requeridas. Os módulos mencionados no capítulo anterior, foram utilizados a partir das necessidades foram surgindo ao decorrer do desenvolvimento da aplicação.

3.2.1 Interface Gráfica com Material-UI e Color Tool

Com Material-UI foi feita toda a parte que o usuário da aplicação tem contato, como os botões, campos de texto. A paleta e cores, primárias e secundárias, foram geradas com a ferramenta Color Tool, que ao escolher as cores, ele lhe dá exemplos de como seria visualmente uma aplicação com estas.

Já com a paleta de cores gerada, com o auxílio do módulo Material-UI, o tema de cores foi criado, como mostra no código a seguir.

```
1 import { createMuiTheme } from '@material-ui/core/styles';
2 const theme = createMuiTheme({
3   palette: {
4     primary: { main: '#00ACC1' },
5     secondary: { main: '#81D4FA' }
6   },
7 //const themeName = 'pacific Blue Malibu Vulture';
8 })
9 export default theme;
```

Listing 3.1 – Criação do tema com paleta de cores secundárias e primárias gerada.

Para utilizar o tema gerado, para cada componente foi feito o *import* do tema (theme) e aplicado com MuiThemeProvider do Material-UI.

```
1 import { MuiThemeProvider } from '@material-ui/core/styles';
2 import theme from '../styles/index.js'
3 ...
4  return (
5       <MuiThemeProvider theme={theme}>
6 ...
```

Listing 3.2 – Utilizando o tema de cores.

3.2.2 Navegação

Para a navegação entre as telas, foi usado o React Router, este além de fazer toda a rota da aplicação, também tornou a aplicação em uma SPA (*Single Page Application*).

Uma SPA significa que ao iniciar a aplicação, todo seu conteúdo é pré-carregado e ao solicitar uma "mudança de página", esta será substituída ou quando alguma ação for requisitada não será necessário recarrega-la totalmente, economizando largura de banda (JADHAV; SAWANT; DESHMUKH, 2015).

3.2.3 Servidores

O servidor da aplicação foi construído com o módulo Express com outros módulos acoplados à ele, como o Mongoose que cuida da comunicação com o banco de dados MongoDB e a modelagem de cada documento/*collection* que equivale a tabelas em banco de dados relacionais.

Também trabalhando junto com o Express o módulo Body Parser que cuida em passar as requisições do cliente para JSON, já que o nosso banco de dados trabalha com esse tipo de documento.

Sem o módulo Cors junto ao Express, a comunicação do Servidor com Cliente ReactJS para envio da UID da *tag* RIFD, não seria possível, gerando vários erros, que são tratados pelo Cors.

O servidor principal da aplicação é construído com o Express, como já dito, mas ao iniciar o servidor da aplicação, é iniciado outros dois servidores: Socket.io, que se comunica com o Socket.io-client em uma das telas que requisita leitura da *tag*; e Net, que é usado para comunicar com o Raspberry, onde é feita a leitura da *tag* RFID.

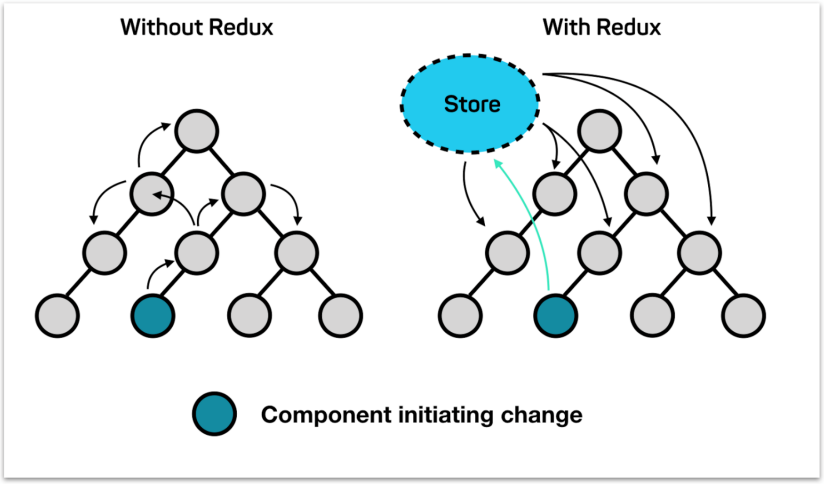
3.2.4 Acessando e Requisitando informações com Redux

Os *states* no React funcionam como variáveis de classe em Java. Cada envio ou solicitação de atualização que se relaciona ao banco de dados, esses *states* são alterados. Neste projeto foi usado o Redux para auxiliar e facilitar esses tipos de ações.

O Redux funciona como uma central de *states*, sendo estes os dados dos usuários e exemplares. Um exemplo de alto nível do funcionamento de uma aplicação, seria a Figura 7. Sem Redux as informações teriam que passar de componente em componente até chegar ao que foi requisitado.

Com o Redux o componente que requisitou informação, por exemplo do usuário atualmente logado no sistema, todas as suas informações já estarão salvas no "*Store*", assim qualquer outro componente poderá acessá-las de modo que não precise de sempre requisitar uma operação **GET** ao banco de dados ou herdar de outro componente.

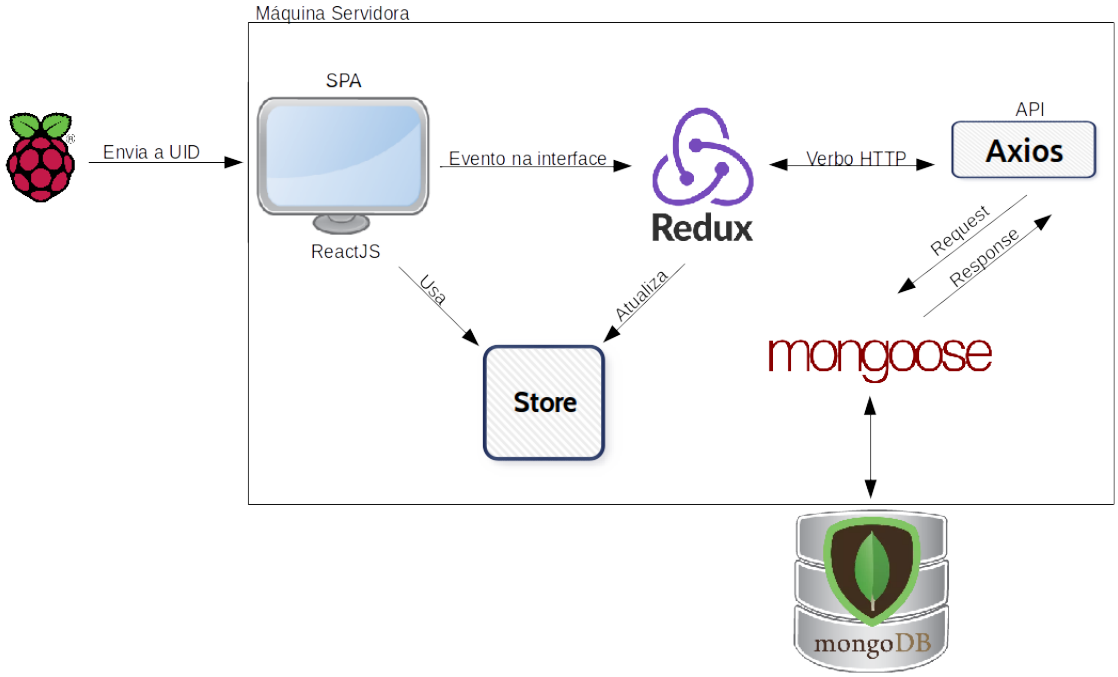
Figura 7 – Sem o uso do Redux e com uso do Redux em uma aplicação ReactJS.



Fonte – Weck (2017)

A Figura 8 mostra o fluxo de dados quando a SPA recebe a UID da *tag* do livro. Ao acontecer um evento de leitura da *tag*, a SPA comunica com o Redux, que passa os dados ao Axios para fazer requisições REST ao banco de dados, este tendo o Mongoose como mediador na comunicação. Ao receber os dados do banco de dados pelo Axios, o Redux atualiza os *states* no Store, para que a SPA ReactJS possa usa-los.

Figura 8 – Arquitetura da Aplicação utilizando Redux.



Fonte – Elaborado pela Autora.

4 PROJETO E DESENVOLVIMENTO

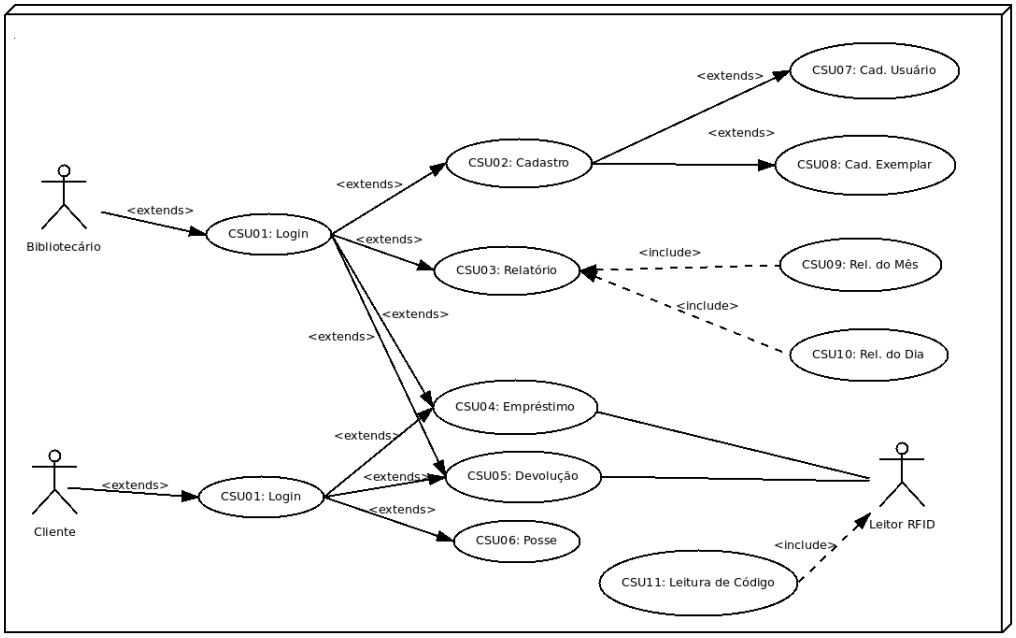
Neste capítulo é descrito a modelagem do projeto - definição das funcionalidades da aplicação, qual o encargo de cada uma delas, o ambiente de desenvolvimento utilizado, manipulação dos dados e como cada material descrito no capítulo anterior foi utilizado. Depois de definida a abordagem, foi feito o desenvolvimento da aplicação.

Por ser um sistema voltado para bibliotecas, lugar onde se tem um grande acervo sobre informações, sejam elas cientificas ou culturais, foi escolhido o nome ***Sophia***, em grego $\Sigma o \varphi \iota \alpha$, que significa ciência e sabedoria[1].

4.1 Modelagem

Antes de começar a programação da aplicação em si, é necessário planejar o que realmente será preciso para o funcionamento correto da aplicação. Usando o software *Dia*, foi feito então o Diagrama de Caso de Uso (CSU) do sistema, como pode ser visto na Figura 9 (sua expansão se encontra no Apêndice A). Conforme desenvolvimento do projeto e amadurecimento do planejamento o diagrama sofreu várias alterações, onde alguns CSU foram removidos, alterados e adicionados.

Figura 9 – Diagrama de Caso de Uso do sistema.



Fonte – Elaborado pela autora.

[1] Sofia - Dicionário Online Dicio <https://www.dicio.com.br/sofia/>

Diagramas de caso de uso organizam os comportamentos do sistema. Ele mostra a relação que há entre os atores e os CSUs (BOOCH; RUMBAUGH; JACOBSON, 2006). Atores são aqueles que irão usufruir das funcionalidades, neste caso temos três atores, as pessoas, Cliente e Bibliotecário, e o leitor RFID.

4.2 Ambientes de Desenvolvimento

Para o desenvolvimento do projeto foi usado o *software* Sublime Text 3, para a codificação da aplicação, em uma máquina com a configurações descritas na Figura 10.

Figura 10 – Configurações do computador pessoal usado.

Fonte – Elaborado pela autora.

Já o Raspberry Pi usado neste mesmo projeto, é o modelo B de sua terceira geração que dispõe da configuração representada a seguir:

- Processador - Broadcom BCM2837;
- CPU - Quad Core 64bit;
- Clock - 1.2GHz;
- RAM - 1GB;
- Wi-Fi - 802.11.b/g/n;
- Bluetooth 4.1;
- GPIO - 40 pinos;
- USB - 4 portas 2.0;
- HDMI.

4.2.1 Raspbian

A aquisição do sistema operacional utilizado foi feita usando o NOOBS[2], 3.0.0, que está disponível gratuitamente no site da organização - anteriormente foi utilizado SO Jessie, que não obteve sucesso à partir das necessidades requisitadas para a comunicação entre GPIO com módulo RC522. NOOBS (*New Out Of Box Software*) é gerenciador de instalação do Raspberry Pi. Além de oferecer o SO *Raspbian*, também disponibiliza a sua versão *Lite*, um emulador de console retro, um *media center* de código aberto, entre outros.

Com a aquisição do NOOBS em formato .zip, foi utilizado um cartão de memória de 8GB, este sendo a recomendação mínima para instalação. Com o cartão em branco, NOOBS descompactado foi passado para o SD e feita a instalação do *Raspbian*.

Raspbian é o sistema operacional oficial da fundação e já vem pré-instalado com software voltados para educação, programação e de uso geral (RASPBERRYPI, 2019).

4.3 Leitura RFID

Para a leitura da *tag* RFID foi determinado o uso do Raspberry e do módulo RC522. A escolha do RPi em detrimento ao Arduino foi pelo fato de ser apenas um microcontrolador não aceitando um sistema operacional como Raspberry. Essa característica no mini computador deixa a vantagem de que a aplicação pode rodar nela, apesar de que não foi aplicada essa opção neste presente projeto, mas a escolha foi feita pensando nesta possibilidade.

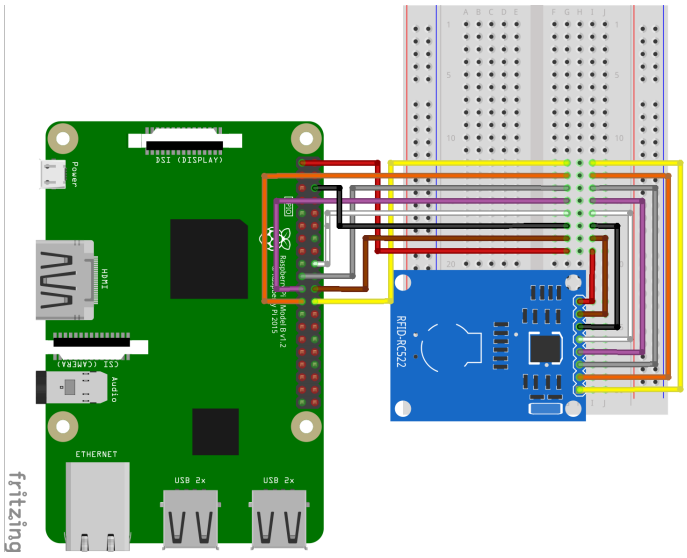
A comunicação entre eles é realizada com a tecnologia SPI, Figura 11, que tem como características ser serial e síncrona. Uma interface periférica serial, que tem um protocolo com quatro linhas de sinal (SCLK, SSn, MOSI, MISO) (LEENS, 2009). O processador tem papel de *master*, através do SCLK, ao transferir dados, que acontece sobre os sinais MOSI e MISO em *full-duplex*, e a seleção do periférico feita sobre o sinal SSn (MARCIEL, 2014).

Figura 11 – Comunicação entre Raspberry, RFID e Pyhton.

Fonte: Imagem retirada do site Embarcados[3]

[2] NOOBS - <https://www.raspberrypi.org/downloads/noobs/>
[3] Embarcados Tecnologia LTDA - <https://www.embarcados.com.br/rfid-raspberry-pi-python/>

Figura 12 – Montagem em ambiente Fritzing do circuito utilizado para a leitura da Tag RFID.



Fonte – Elaborado pela Autora.

O código e a montagem utilizados para fazer a leitura da *tag* RFID foram baseadas no repositório no GITHUB[4] de Ondryáš (2018). A conexão entre os pinos do RC522 e o Raspberry foi feita de acordo com a Figura 12 seguindo a pinagem descrita na Tabela 1.

Tabela 1 – Pinagem seguida para a montagem

RC522 (Nome/Nº)		Raspberry Pi 3 (Nome/Nº)	
SDA	1	GPIO8	24
SCK	2	GPIO11	23
MOSI	3	GPIO10	19
MISO	4	GPIO9	21
IRQ	5	GPIO24	18
GND	6	Ground	6
RST	7	GPIO25	22
3.V	8	3V3	1

Fonte – Elaborado pela autora.

O código de leitura da *tag* RFID utiliza a biblioteca *pirc522*. Para envio do UID (*Unique Identifier*) RFID foi desenvolvida um algoritmo que utiliza comunicação via *socket*, responsável por enviar identificador único da *tag* para o servidor localizado em outro computador, neste caso com IP `10.120.253.102`.

```
1 from pirc522 import RFID
2 import socket
3 import sys
4 import time
```

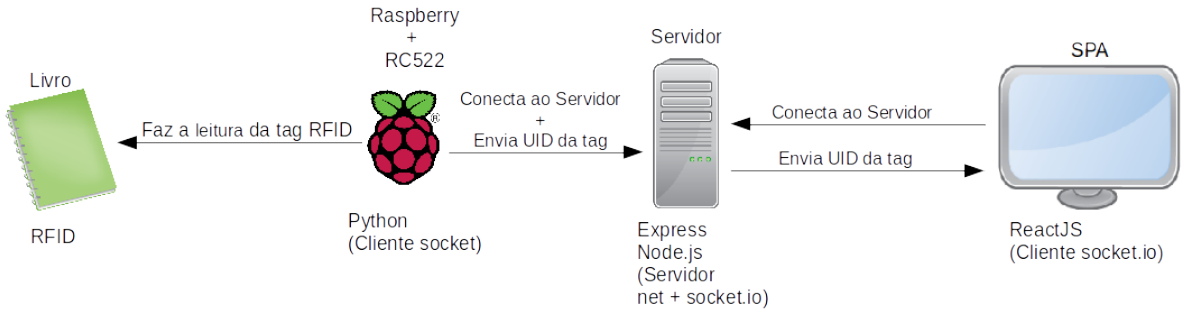
[4] https://github.com/ondryaso/pi-rc522

```
5  rdr = RFID()
6  while True:
7      # Espera pela aproximacao da tag
8      rdr.wait_for_tag()
9      (error, tag_type) = rdr.request()
10     if not error:
11         (error, uid) = rdr.anticoll()
12         if not error:
13             # imprimi UID da tag
14             print(str(uid))
15             # cria um socket para o envio da UID
16             sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17             # Coneta o socket ao endereco e porto
18             server_address = ('10.120.253.102', 29298)
19             sock.connect(server_address)
20             try:
21                 # Manda a UID para o servidor
22                 message = str(uid)
23                 sock.sendall(message)
24             finally:
25                 # fecha conexao
26                 sock.close()
27             time.sleep(5)
```

Listing 4.1 – Código em python para a leitura da *tag* RFID e envio de sua UID.

Assim que o servidor é iniciado, ele aguarda por uma conexão do cliente python, ou seja, espera até que o usuário aproxime uma *tag* para a leitura da UID no Raspberry. Do mesmo modo o cliente SPA se conecta ao Servidor para receber a informação da *tag*. Esta comunicação é representada pela Figura 13. Tanto *backend* quanto o *frontend* se encontram na mesma máquina servidora.

Figura 13 – Arquitetura de comunicação entre o servidor, aplicação ReactJs e Raspberry Pi 3 com módulo RC522.



Fonte – Elaborado pela autora.

Um exemplo do funcionamento desta comunicação seria ao usuário estar logado

e solicitar um empréstimo. Ao requisitar adicionar um livro, a leitura da *tag* do livro será acionada. Ao fazer a leitura, o cliente pyhton enviará a UID ao servidor Net na máquina Servidora que se encontra também a aplicação, guardando esta informação em uma variável que o servidor Socket.io também tenha acesso, e repassa para o cliente Socket.io na aplicação que mostra a UID em um campo de texto.

4.4 Manipulação dos dados

Dada a utilização do MongoDB como banco de dados do projeto, foi preciso fazer uso da biblioteca Mongoose para a comunicação entre a aplicação e o MongoDB, já que o banco de dados não se comunica diretamente com a aplicação. O trecho de código a seguir mostra como foi feita a conexão com MongoDB.

```
1 const mongoose = require('mongoose');
2 /* mongodbUri = 'mongodb://localhost:27017/bib'*/
3 mongoose.connect(config.mongodbUri, { useNewUrlParser: true })
4     .then(() => console.log('Banco de Dados Conectado'))
5     .catch((err) => console.log('Erro na conexao com banco de dados',
          err));
6 /* Para evitar a depreciacao do MongoDb */
7 mongoose.set('useFindAndModify', false);
```

Listing 4.2 – Código em JavaScript para conexão com banco de dados MongoDB usando a biblioteca Mongoose do Node.js.

Cada exemplar teria um documento detalhando como seria armazenados seus dados, que foi especificado usando a função *Schema* do Mongoose, como no código listado a seguir.

```
1  const DataExe = new Schema({
2    exe_RFID: String,
3    exe_Titulo: String,
4    exe_SubTitulo: String,
5    exe_Autor: String,
6    exe_Edicao: Number,
7    exe_Editora: String,
8    exe_NumPaginas: Number,
9    exe_Ano: Number,
10   exe_ISBN: String,
11   exe_Emprestado: Boolean,
12   exe_Historico:[ {
13     usuID: String,
14     dataHoraEmp: Date,
15     dataHoraDev: Date
16   }]
17 },{
```

```
18   collection: 'DataExe'
19 });
```

Listing 4.3 – *Schema* referente ao documento de Exemplar.

O `exe_RFID` seria o UID da *tag* de cada exemplar, `exe_Titulo` o seu nome, `exe_Subtitulo` se houver subtitulo, `exe_Autor` nome do(s) autor(es), `exe_Edicao` será a versão do exemplar em número, `exe_Editora` o nome da editora que o publicou, `exe_ISBN` é a "identidade"do exemplar, `exe_NumPaginas` número total de páginas, `exe_Ano` o ano em que foi lançado, `exe_Emprestado` será manipulado em caso o livro seja emprestado, *True* caso esteja emprestado e *False* esteja na biblioteca, e `exe_Historico` contendo a ID do usuário que já teve o livro em posse, juntamente com a data de quando o pegou emprestado e quando devolveu.

No caso do usuário, este terá suas informações pessoais armazenadas, tais como nome, data de nascimento, CPF, endereço, etc. O campo `usu_Admin` pode assumir os valores *True*, se for um administrador do sistema (Bibliotecário com permissões administrativas), ou *False* no caso de um usuário da biblioteca. Já `usu_Bloqueado` terá seu valor trocado para *True* se o usuário não devolver o material na data limite de devolução. O campo `usu_ExemplarPosse` armazena o ID do exemplar que o cliente está em posse no momento, juntamente com a data em que deve ser devolvido à biblioteca. Por fim, `usu_Historico` tem os campos de data em que foi obtido o empréstimo, data da devolução e a ID do exemplar.

```
1  const DataUsu = new Schema({
2    usu_Nome: String,
3    usu_Nasc: Date,
4    usu_Tel: String,
5    usu_CPF: { type: String, required: true, unique: true },
6    usu_Endereco: String,
7    usu_Bairro: String,
8    usu_Cidade: String,
9    usu_Estado: String,
10   usu_Mae: String,
11   usu_Email: String,
12   usu_Admin: Boolean,
13   usu_Bloqueado: Boolean,
14   usu_Senha: { type: String, required: true },
15   usu_PosseQuant: {type: Number, min: 0, max: 3},
16   usu_ExemplarPosse: [{
17     exeID: String,
18     devolucao: Date
19   }],
20   usu_Historico: [{
21     dataHoraEmp: Date,
22     dataHoraDev: Date,
```

```
    exeID: String
  }]
},{
  collection: 'DataUsu'
});
```

Listing 4.4 – *Schema* referente ao documento de Usuário.

4.4.1 Operações da API REST

Para fazer as operações CRUD, foi usado o módulo *axios* para fazer as operações REST (`post, get, put e delete`) equivalentes. Foi feito um código para cada tipo de requisição ao banco de dados.

```
export const createUsu = ({ usu_Nome, usu_Mae, usu_CPF, usu_Nasc,
   usu_Tel, usu_Endereco, usu_Bairro, usu_Cidade, usu_Estado, usu_Email,
    usu_Admin, usu_Bloqueado, usu_Senha }) => {
  return (dispatch) => {
    return axios.post(`${apiUrlUsu}`, { usu_Nome, usu_Mae, usu_CPF,
       usu_Nasc, usu_Tel, usu_Endereco, usu_Bairro, usu_Cidade,
       usu_Estado, usu_Email, usu_Admin, usu_Bloqueado, usu_Senha })
      .then(response => {
        dispatch(createUsuSuccess(response.data))
      })
      .catch(error => {
        throw(error);
      });
  };
};
```

Listing 4.5 – Código em JavaScript para cadastro do usuário via post.

A constante `createUsu` cuida da criação do usuário por meio do `post`, com a entrada dos dados requiridos na hora do cadastro, caso não haja nenhum tipo de erro o `createUsuSuccess` retorna os dados de acordo como foram armazenados. Do mesmo modo foi feito ao alterar o usuário, com a diferença no comando `put` e acrescentado o `_id` para encontrar o usuário.

```
export function updateUsu({ _id, usu_Nome, usu_Mae, usu_CPF, usu_Nasc,
   usu_Tel, usu_Endereco, usu_Bairro, usu_Cidade, usu_Estado, usu_Email,
    usu_Admin, usu_Bloqueado, usu_Senha, usu_ExemplarPosse,
   usu_Historico }) {
  return (dispatch) => {
    return axios.put(`${apiUrlUsu}/${_id}`, { usu_Nome, usu_Mae, usu_CPF
       , usu_Nasc, usu_Tel, usu_Endereco, usu_Bairro, usu_Cidade,
       usu_Estado, usu_Email, usu_Admin, usu_Bloqueado, usu_Senha,
       usu_ExemplarPosse, usu_Historico })
      .then(response => {
```

```
5             dispatch(updateUsuSuccess(respose.data));
6         })
7         .catch(error => {
8             throw (error);
9         });
10    };
11 };
```

Listing 4.6 – Código em JavaScript para atualização de dados do usuário via put.

```
1  export const deleteUsu = id => {
2    return (dispatch) => {
3      return axios.delete(`${apiUrlUsu}/${id}`)
4        .then(response => {
5          dispatch(deleteUsuSuccess(response.data))
6        })
7        .catch(error => {
8          throw(error);
9        });
10   };
11 };
```

Listing 4.7 – Código em JavaScript para deleção de um usuário em específico via delete.

Ao deletar o Usuário é necessário apenas a identificação para remover um documento específico do banco de dados, o método `fetchAllUsus` pega todos os usuários cadastrados, caso não haja nenhum, retorna um *Array* de tamanho 0.

```
1  export const fetchAllUsus = () => {
2    return (dispatch) => {
3      return axios.get(apiUrlUsu)
4        .then(response => {
5          dispatch(fetchUsus(response.data))
6        })
7        .catch(error => {
8          throw(error);
9        });
10   };
11 };
```

Listing 4.8 – Código em JavaScript para obtenção de todos usuários cadastrados via get.

Os métodos acima foram replicados para manipulação dos dados de exemplar, com os parâmetros de acordo com registro do livro.

A Figura 14 mostra o comportamento de um documento do tipo usuário ao cadastrar um usuário, 14a, e como fica após realizar empréstimo de um livro, 14b, `exeID` se refere á *tag* RFID do livro. Do mesmo modo, campo `exe_Emprestado` do exemplar referente trocará seu valor para *true*. Como ao realizar o empréstimo o campo `usu_ExemplarPosse`

Figura 14 – Representação de um documento do tipo Usuário.

(a) Criação de um documento com `post`.


```
{
    "_id": "5ceacced2552e33279475c5e",
    "usu_Nome": "José da Silva",
    "usu_Mae": "Maria da Silva",
    "usu_CPF": "452.071.323-42",
    "usu_Nasc": "1985-04-19T00:00:00.000Z",
    "usu_Tel": "(37)9568-6254",
    "usu_Endereco": "Rua H",
    "usu_Bairro": "Norte",
    "usu_Cidade": "Aracaju",
    "usu_Estado": "Sergipe",
    "usu_Email": "josedasilva@gmail.com",
    "usu_Admin": false,
    "usu_Bloqueado": false,
    "usu_Senha": "#trend@245!if",
    "usu_ExemplarPosse": [],
    "usu_Historico": [],
    "__v": 0
}
```


(b) Alteração com `put` apos realizar empréstimo de um livro.


```
    "_id": "5ceacced2552e33279475c5e",
    "usu_Nome": "José da Silva",
    "usu_Mae": "Maria da Silva",
    "usu_CPF": "452.071.323-42",
    "usu_Nasc": "1985-04-19T00:00:00.000Z",
    "usu_Tel": "(37)9568-6254",
    "usu_Endereco": "Rua H",
    "usu_Bairro": "Norte",
    "usu_Cidade": "Aracaju",
    "usu_Estado": "Sergipe",
    "usu_Email": "josedasilva@gmail.com",
    "usu_Admin": true,
    "usu_Bloqueado": true,
    "usu_Senha": "#trend@245!if",
    "usu_ExemplarPosse": [
        {
            "_id": "5cead5f31c8cee35417434c5",
            "exeID": "[98, 35, 154, 31, 196]",
            "devolucao": "2019-04-19T00:00:00.000Z"
        }
    ],
    "usu_Historico": [],
```


Fonte – Elaborado pela autora.

sofre alterações, o `usu_Historico` do usuário e `exe_Historico` do exemplar também mudarão ao realizar a devolução.

4.5 Considerações do Desenvolvimento

Neste capítulo foram apresentados o desenvolvimento e modelagem do projeto. No próximo capítulo serão apresentados os resultados do protótipo desenvolvido.

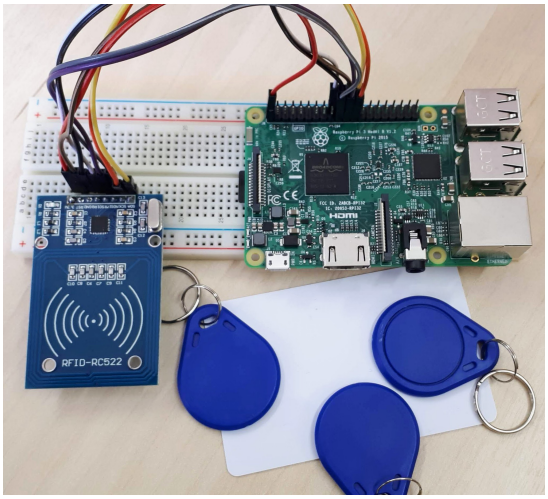
5 RESULTADOS

Neste capítulo será apresentado os resultados de acordo com o que foi apresentado no capítulo anterior.

5.1 Etiquetas RFID e circuitos

Como descrito no capítulo anterior, para a leitura da *tag* RFID foi elaborado um circuito usando o módulo RC522 e Raspberry Pi 3, como pode ser visto na Figura 15, tanto o cartão quanto os chaveiros são *tags*, estas foram usadas para simular a leitura da *tag* de um livro.

Figura 15 – Montagem do circuito real em *protoboard*.



Fonte – Elaborado pela autora.

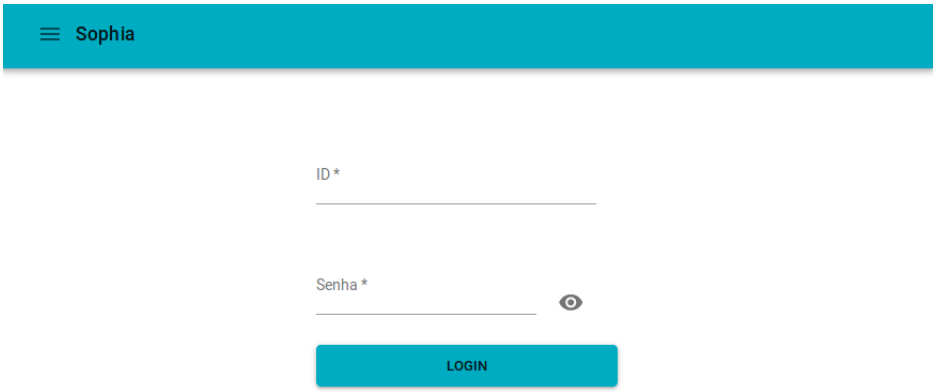
5.2 GUI - Interface Gráfica de Usuário

As cores do sistemas, tons de azul, também foram escolhidas propositalmente, pela cor com tons de azul claro trazer "calma". A paleta de cores foi gerada com a ferramente Color Tool, descrita no Capítulo 3.

Tela de Login, Figura 16, é a visão inicial do sistema, onde o usuário já cadastrado pode entrar com seus dados, ID e senha. No campo de senha, tem a opção de visualizá-la ou mantê-la escondida (através do ícone que parece um olho), a segunda sendo o padrão.

Ao logar no sistema, o usuário é redirecionado para tela principal, a tela Home. Ela oferece as opções principais: empréstimo, devolução, posse, cadastro e relatórios (sendo as

Figura 16 – Tela Login.



Fonte – Elaborado pela autora.

duas últimas opções para administradores). Na Figura 17, é disposta perspectiva de um cliente comum (que não há permissões administrativas) ao sistema Sophia.

Figura 17 – Tela Home, a principal tela da aplicação.

Fonte – Elaborado pela autora.

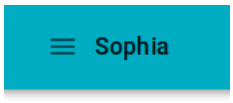
Na barra de ferramentas, que se encontra na parte superior da tela, existe o nome da aplicação e um botão destinado ao menu do sistema, 18b. Ao clicar neste botão, é exibido um leque de opção para o usuário, como pode ser visto na 18a. As opções de `cadastro` e `relatório` podem ser expandidas, como está na Figura 18(a), ou contidas.

Ao escolher a opção de empréstimo, ao clicar no botão de adicionar (simbolizado com '+'), é aberta uma janela de diálogo mostrando uma mensagem para o usuário aproximar a tag do exemplar ao leitor ou digitar o código etiquetado no livro, Figura 19, esperando até que uma ação seja feita.

Figura 18 – Menu lateral.

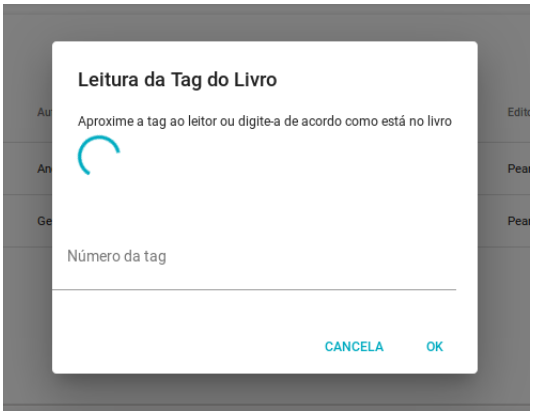
(a) Expansão do menu lateral.

(b) Botão do Menu Lateral na barra de ferramentas.



Fonte – Elaborado pela autora.

Figura 19 – Diálogo de espera de leitura de RFID ou entrada por teclado.

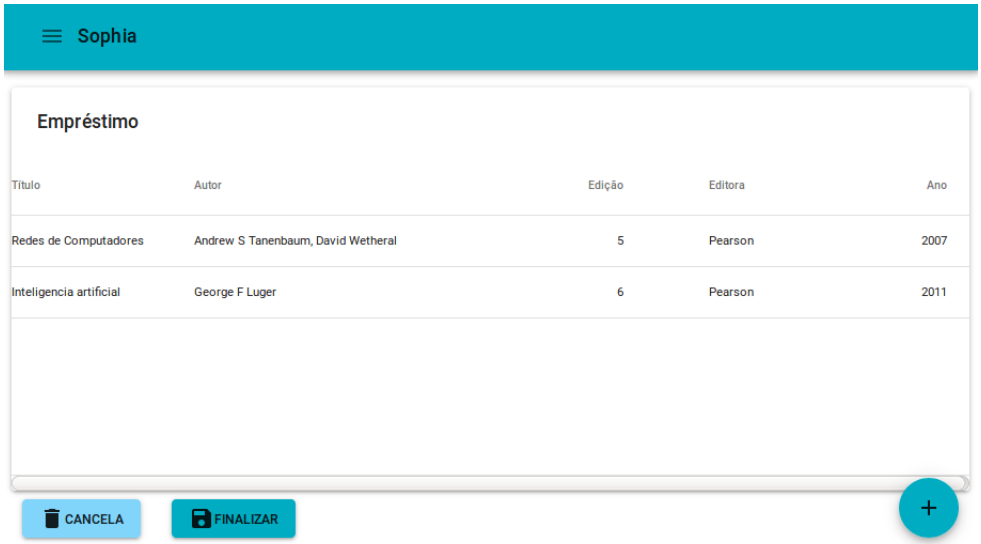


Fonte – Autora.

Ao adicionar itens á lista, o usuário tem um resultado parecido com a Figura 20, tendo a opção de cancelar ou finalizar a ação de empréstimo.

Devolver os exemplares funciona como o empréstimo, mas com a diferença de apenas o bibliotecário ter a opção de digitar o código do livro, como uma forma de garantia.

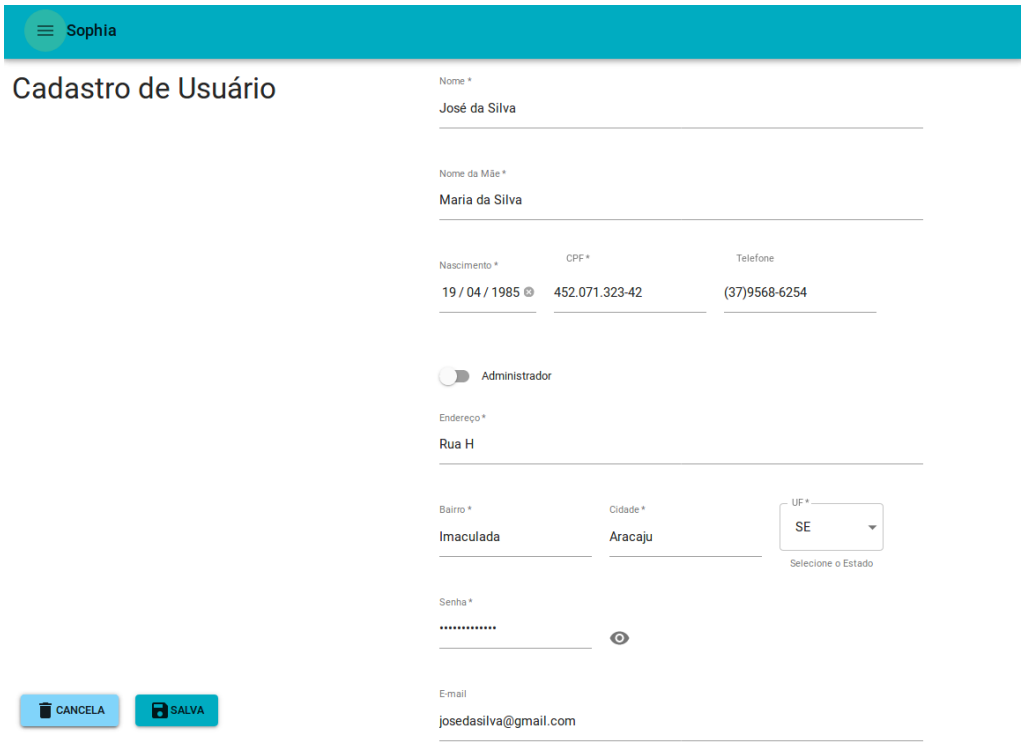
Figura 20 – Tela de Empréstimo, já com itens na lista.



Fonte – Elaborado pela autora.

Ao cadastrar um usuário, o administrador entra com todos os dados requeridos. Os campos de CPF e Telefone possuem máscaras, por serem tipos de dados com um padrão. E a opção de `administrador` estiver verificada, o usuário terá permissões de administrador.

Figura 21 – Tela de cadastro de usuário

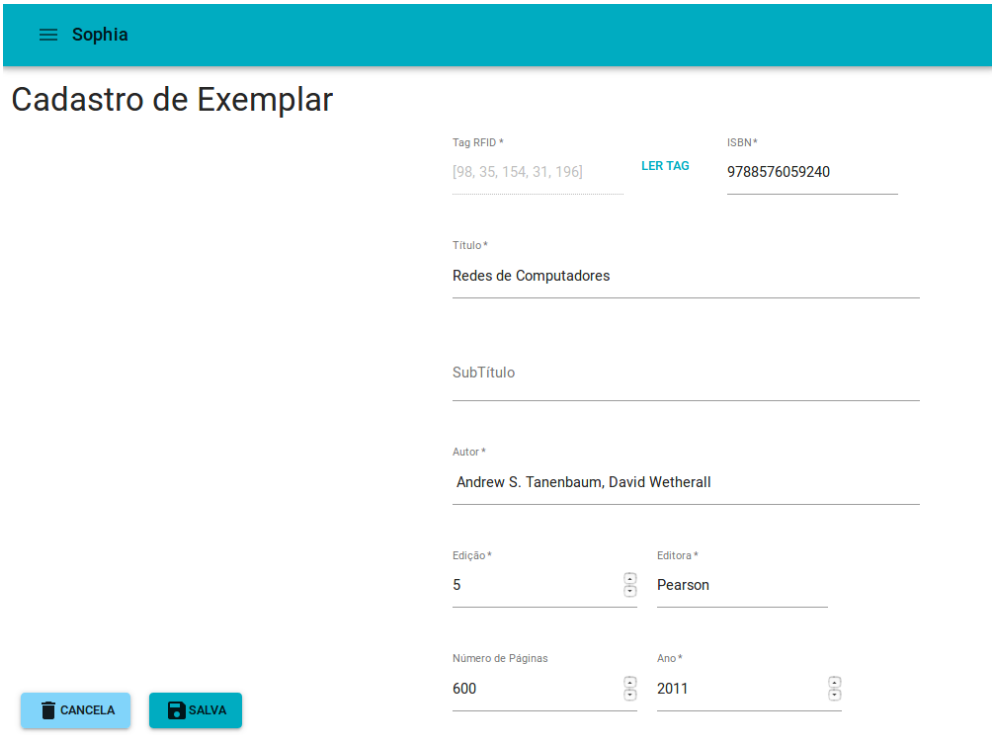


Fonte – Elaborado pela autora.

Por fim, como no cadastro de usuário, ao cadastrar um exemplar o bibliotecário

precisa entrar com todos os dados requeridos. O botão “Ler Tag” é destinado para a leitura da etiqueta RFID.

Figura 22 – Tela de cadastro de exemplares já com campos preenchidos com informações do livro a ser inserido.



Fonte – Elaborado pela autora.

5.3 Consideração sobre os resultados do protótipo

Neste capítulo foram apresentados os resultados do protótipo desenvolvido. O sistema é capaz de cadastrar livros e usuários, realizar o empréstimo, consulta e devolução de exemplares para usuários de forma automatizada.

O protótipo apresentado é versátil o bastante para agregar diversas outras funcionalidades, o que é possível graças às tecnologias e *frameworks* utilizados. Por fim, a utilização de tecnologias de baixo custo, como o Raspberry e sensores RFID, e softwares livres[1] viabiliza a instalação do sistema Sophia em qualquer ambiente bibliotecário.

[1] Software livre é o software que concede liberdade ao usuário para executar, acessar e modificar o código fonte, e redistribuir cópias com ou sem modificações.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, foi desenvolvido e apresentado um sistema de automatização bibliotecário para o processo de empréstimo e devolução de livros, denominado Sophia.

O protótipo apresentado é capaz de cadastrar usuários e exemplares, realizar empréstimos e devoluções, e outras funcionalidades. Além disto, possui uma interface leve e intuitiva, com elementos dispostos de modo a facilitar a utilização do sistema por bibliotecários e clientes da biblioteca. Além disso, o sistema proposto tem a característica de autoatendimento, o qual é um diferencial entre os sistemas similares citados no Capítulo 2, que não possuem este tipo de funcionalidade.

É importante frisar que o objetivo do sistema não é substituir o papel do bibliotecário. Longe disto, o sistema funciona como uma ferramenta para auxiliar o seu trabalho, agilizando no processo de empréstimo e devolução de exemplares por parte dos usuários. Desta forma, o funcionário poderá focar em tarefas pouco triviais as quais concerne à sua formação, como por exemplo administrar informações, desenvolver coleções de documentos, auxiliar usuários a encontrar informações em registros, catalogar documentos, etc.

A utilização do Raspberry Pi neste trabalho para a leitura das *tags* foi proposital, em questão de que em trabalhos futuros toda aplicação possa ser remanejada para o mini computador. Trabalhos futuros também envolvem integrar algumas funcionalidades no sistema, como por exemplo login com cartão RFID do cliente e/ou empréstimo de modo automático sem que o cliente interaja com a aplicação de forma explícita, ou seja, o cliente apenas pega os itens de seu interesse e a aplicação faz a leitura de forma remota dos livros assim que o usuário deixa o local. Tecnologias como esta, já estão sendo aplicadas em lojas de conveniências da *Amazon* nos EUA, conhecidas como *Amazon Go*[1].

Em relação à garantia de empréstimo, de modo que não seja forjada a operação, à sua garantia não é 100%, pois não foi possível integrar uma antena - geralmente encontradas em estabelecimentos que usam RFID ou similares, para controle de furto - ficando apto sua integração à aplicação para trabalhos futuros, assim como o cancelamento unitário, após adicionar os livros em que tem interesse.

Mais uma possibilidade, seria o comprovante de empréstimo enviado para e-mail e/ou SMS, *Whatsapp*, como uma forma de notificar ao usuário de empréstimo, evitando o atraso na devolução, e até mesmo acesso indevidos à sua conta.

Apesar da complexidade do projeto desenvolvido dado aos vários *frameworks* utilizados, este projeto contribui na área de aplicações voltadas para Web para a comunidade,

[1] <https://amzn.to/2haWZhx>

bem como para sistemas bibliotecários automatizados. Além do mais, esta monografia apresentou todo poder que tecnologias JavaScript podem oferecer para um desenvolvimento de aplicações em geral, sejam voltadas para *desktops*, sistemas de autoatendimento ou até mesmo dispositivos móveis.

REFERÊNCIAS

ARDUINO. *Arduino*. 2019. Acessado em: 20 de Jun. 2019. Disponível em: <https://bit.ly/2wXrlck>. Citado na página 34.

AXIOS, M. *Promise based HTTP client for the browser and node.js*. 2019. Acessado em: 22 Abr. de 2019. Disponível em: <https://github.com/axios/axios>. Citado na página 31.

BANKS, A.; PORCELLO, E. *Learning React: Functional Web Development with React and Redux*. [S.l.]: "O'Reilly Media, Inc.", 2017. Citado 2 vezes nas páginas 29 e 30.

BERA, M. H. G.; MINE, A. F.; LOPES, L. F. B. Mean stack: Desenvolvendo aplicações web utilizando tecnologias baseadas em javascript. 2015. Citado na página 31.

BOAGLIO, F. *MongoDB: construa novas aplicações com novas tecnologias*. [S.l.]: Editora Casa do Código, 2017. Citado na página 33.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. [S.l.]: Elsevier Brasil, 2006. Citado na página 42.

CALÇADO, V. L. X. d. S. *Influência da Utilização de Processo Unificado, Testes e Métricas na Qualidade de Produtos de Software*. Tese (Doutorado) — UNIVERSIDADE DE BRASÍLIA, 2007. Citado na página 37.

CODE, V. S. Node.js. 2009. Citado na página 31.

COMPONTENTS styled. *styled-components*. 2016. Acessado em: 25 Jan. de 2019. Disponível em: <https://www.styled-components.com/>. Citado na página 30.

CROCKFORD, D. Javascript: The world's most misunderstood programming language. 2001. Citado na página 28.

CROCKFORD, D. *The application/json media type for javascript object notation (json)*. [S.l.], 2006. Citado na página 33.

DAMASIO, E.; RIBEIRO, C. Software livre para bibliotecas, sua importância e utilização: o caso gnuteca. *Revista Digital de Biblioteconomia e Ciencia da Informação*, UNICAMP-Biblioteca Central, v. 4, n. 1, 2006. Citado na página 26.

DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004. Citado 2 vezes nas páginas 31 e 32.

DB-ENGINES. *DB-Engines Ranking*. 2019. Disponível em: <https://db-engines.com/en/ranking>. Citado na página 33.

DIAS, G. A.; SILVA, M. B. d. O sistema de automacão em bibliotecas openbiblio aplicado à disciplina automacão em bibliotecas. *Biblionline*, v. 6, n. 1, p. 53–71, 2010. Citado na página 27.

DIAS, R. R. G.; MIRANDA, W. Software livre: Potencial de utilização no âmbito acadêmico. 2013. Citado na página 26.

EXPRESSJS. *Express - framework de aplicativo da web Node.js*. Acessado em: 8 de Mai. de 2019. Disponível em: <https://expressjs.com/pt-br/>. Citado na página 30.

EXPRESSJS, M. *Middleware Node.js CORS*. 2019. Acessado em: 1 Mai. 2019. Disponível em: <https://github.com/expressjs/cors>. Citado na página 31.

FINKENZELLER, K. *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. [S.l.]: John Wiley & Sons, 2010. Citado na página 35.

FLANAGAN, D. *JavaScript: O guia definitivo*. [S.l.]: Bookman Editora, 2004. Citado 2 vezes nas páginas 27 e 28.

GOOGLE, I. *Color Tool - Material Design*. 2019. Acessado em: 9 de Mai de 2019. Disponível em: <https://material.io/tools/color/>. Citado na página 29.

HAHN, E. *Express in Action: Writing, building, and testing Node. js applications*. [S.l.]: Manning Publications„ 2016. Citado na página 30.

HOLMES, S. *MEAN Definitivo com Mongo, Express, Angular e Node*. [S.l.]: Novatec Editora, 2016. Citado 5 vezes nas páginas 27, 30, 31, 32 e 34.

INFOISIS. *CDS/ISIS*. 2004. Acessado em: 30 de Abr. de 2019. Disponível em: <http://www.infoisis.com.br/html/cdsisis.html>. Citado na página 26.

JACQUES, A. *Entendendo o CORS*. 2016. Acessado em: 22 de Mar. 2019. Disponível em: <https://bit.ly/2VOj6Pe>. Citado na página 31.

JADHAV, M. A.; SAWANT, B. R.; DESHMUKH, A. Single page application using angularjs. *International Journal of Computer Science and Information Technologies*, Citeseer, v. 6, n. 3, p. 2876–2879, 2015. Citado na página 39.

JOEL. Javascript tutorial: Introdução ao desenvolvimento web. DevMedia, 2016. Acessado em: 25 de Jun. de 2019. Citado na página 21.

KOPPALA, J. Erp solution with reactjs. Metropolia Ammattikorkeakoulu, 2018. Citado na página 28.

LANDT, J. The history of rfid. *IEEE potentials*, IEEE, v. 24, n. 4, p. 8–11, 2005. Citado na página 35.

LEENS, F. An introduction to i 2 c and spi protocols. *IEEE Instrumentation & Measurement Magazine*, IEEE, v. 12, n. 1, p. 8–13, 2009. Citado na página 43.

MAKINO, A. Abordagem da metodologia rup no desenvolvimento de um sistema de gestão comercial. 2009. Citado na página 37.

MARCIEL, V. *Comunicação SPI em Linux*. 2014. Acessado em: 13 de Mai. de 2019. Disponível em: <https://bit.ly/2VZbybJ>. Citado na página 43.

MATERIAL-UI. *The world's most popular React UI framework - Material UI*. 2019. Acessado em: 8 de Mai. 2019. Disponível em: <https://material-ui.com/>. Citado na página 29.

MIKI, H. Micro-isis: uma ferramenta para o gerenciamento de bases de dados bibliográficas. *Ciência da Informação*, v. 18, n. 1, 1989. Citado na página 26.

MILANESI, L. *Biblioteca*. [S.l.]: Ateliê editorial, 2002. Citado na página 26.

MONGODB, I. *MongoDB*. 2019. Acessado em: 2 de Mai. de 2019. Disponível em: <https://www.mongodb.com>. Citado na página 33.

MONGOOGSEJS. *Mongoose ODM v5.5.6*. 2011. Acessado em: 8 de Mai. de 2019. Disponível em: <https://mongoosejs.com/>. Citado na página 31.

OBJETOS, O. à; JUNIOR, G. B. ao processo unificado. *Tradução Luiz Augusto Meirelles Salgado e João*, 2004. Citado 2 vezes nas páginas 37 e 38.

ONDRYáš, O. *Raspberry Pi Python library for SPI RFID RC522 module*. 2018. Acessado em: 17 de Jun. 2019. Disponível em: <https://github.com/ondryaso/pi-rc522>. Citado na página 44.

OPENBIBLIO. *OpenBiblio*. 2017. Acessado em: 2 de Mai. de 2019. Disponível em: <http://obiblio.sourceforge.net/>. Citado na página 27.

PIRES, J. *O que é API? REST e RESTful? Conheça as definições e diferenças!* 2017. Acessado em: 21 Ago. de 2019. Disponível em: <https://bit.ly/2VyNdoT>. Citado na página 34.

POSTMAN. *Postman*. 2019. Acessado em: 20 Ago. de 2018. Disponível em: <https://www.getpostman.com/postman>. Citado na página 34.

POWERS, S. *Aprendendo Node: Usando JavaScript no Servidor*. [S.l.]: "Novatec Editora LTDA ", 2017. Citado na página 27.

RASPBERRYPI, F. *Teach, Learn And Make With Raspberry Pi*. 2019. Acessado em: 2 Fev. 2019. Disponível em: <https://www.raspberrypi.org/>. Citado 2 vezes nas páginas 35 e 43.

REACT. *React - A JavaScript library for building user interfaces*. 2019. Disponível em: <https://reactjs.org/>. Citado 2 vezes nas páginas 28 e 29.

REDUX. *Redux*. 2019. Acessado em: 1 de Mai. de 2019. Disponível em: <https://redux.js.org/>. Citado na página 30.

RIBEIRO, S. *Como Criar uma Aplicação Full-Stack com React*. 2019. Acessado em: 20 de Abr. 2019. Disponível em: <https://bit.ly/2WSe9S0>. Citado 2 vezes nas páginas 30 e 31.

RICHARDSON, M.; WALLACE, S. Primeiros passos com o raspberry pi. *Primeira Edição. Novatec Editora Ltda*, p. 20, 2013. Citado na página 35.

ROBBESTAD, S. A. *ReactJS Blueprints*. [S.l.]: Packt Publishing Ltd, 2016. Citado na página 28.

ROBOCORE. *Kit RFID Mfrc522 (13,56MHz)*. 2018. Acessado em: 19 Dez. 2018. Disponível em: <https://www.robocore.net/loja/wireless/kit-rfid-mfrc522>. Citado na página 36.

RUBENS, J. *Preimeiros passos com Node.js*. [S.l.]: Editora Casa do Código, 2017. Citado 3 vezes nas páginas 27, 30 e 37.

SANTOS, N. M. L. Automação de biblioteca universitária: análise comparativa do software livre gnuteca com o software proprietário pergamum. 2008. Citado na página 21.

SEBESTA, R. W. *Conceitos de linguagens de programação*. [S.l.]: Bookman Editora, 2009. Citado 2 vezes nas páginas 21 e 28.

SIGNIFICADOS. *Significados*. 2019. Acessado em: 18 de Jan. 2019. Disponível em: <https://www.significados.com.br>. Citado na página 21.

SILBERSCHATZ, A.; SUNDARSHAN, S.; KORTH, H. F. *Sistema de banco de dados*. [S.l.]: Elsevier Brasil, 2016. Citado na página 32.

SILVA, E. *Construíndo Aplicativos em Plataforma Cruzada com HTML5*. 2013. Acessado em: 25 de Jun. 2019. Disponível em: <https://bit.ly/2LiXRi8>. Citado na página 25.

SOLIS. *Gnuteca*. 2018. Acessado em: 30 de Abr. de 2019. Disponível em: <https://www.solis.com.br/gnuteca>. Citado na página 26.

SOUTO, M. *Roteamento no React com os poderes do React Router v4*. 2018. Acessado em: 3 de Abr. de 2019. Disponível em: <https://bit.ly/2KHQtsP>. Citado na página 30.

SUBLIME, H. P. L. *Sublime Text*. 2019. Acessado em: 20 de Mai. 2019. Disponível em: <https://www.sublimetext.com/>. Citado na página 25.

TRAINING, R. *React Router: Declarative Routing for React.js*. 2019. Acessado em: 2 de Abr. de 2019. Disponível em: <https://reacttraining.com/react-router/>. Citado na página 30.

UPTON, E.; HALFACREE, G. *Raspberry Pi user guide*. [S.l.]: John Wiley & Sons, 2014. Citado na página 35.

VIPUL, A.; SONPATKI, P. *ReactJS by Example-Building Modern Web Applications with React*. [S.l.]: Packt Publishing Ltd, 2016. Citado na página 28.

VUJOVIĆ, V.; MAKSIMOVIĆ, M. Raspberry pi as a sensor web node for home automation. *Computers & Electrical Engineering*, Elsevier, v. 44, p. 153–171, 2015. Citado na página 35.

W3TECHS. *Comparison of the usage of Angular vs. Vue.js vs. React for websites*. 2019. Acessado em: 2 de Mai. de 2019. Disponível em: <https://w3techs.com/technologies/comparison/js-angularjs,js-react,js-vuejs>. Citado 2 vezes nas páginas 28 e 29.

WANT, R. An introduction to rfid technology. *IEEE pervasive computing*, IEEE, n. 1, p. 25–33, 2006. Citado 2 vezes nas páginas 35 e 36.

WECK, S. *Developing modern offline apps with ReactJS, Redux and Electron – Part 3 – ReactJS + Redux*. 2017. Acessado em: 20 de Jun. 2019. Disponível em: <https://bit.ly/2WTxFNh>. Citado na página 40.

WIKI, G. *Dia*. 2018. Acessado em: 17 de Abr. 2018. Disponível em: <https://wiki.gnome.org/Apps/Dia>. Citado na página 25.

WOOD, L. et al. Document object model (dom) level 1 specification. *W3C recommendation*, v. 1, 1998. Citado na página 28.

Apêndices

APÊNDICE A – EXPANSÃO DE CASO DE USO

CSU01: Login
Sumário O usuário acessa o sistema com seus dados de acesso.
Atores envolvidos Bibliotecário e Cliente.
Pré-condições Cliente e/ou Bibliotecário cadastrados no sistema.
Fluxo principal Usuário insere login e senha.
Fluxos de exceção Se inserir senha ou usuário incorreto o acesso será negado.
Fluxos alternativos N/A.
Pós-condições O usuário terá acesso ao sistema conforme o seu tipo de usuário.

CSU02: Cadastro
Sumário O bibliotecário realiza todos os cadastros de usuário, calendário.
Atores envolvidos Bibliotecário.
Pré-condições O bibliotecário deve estar logado no sistema.
Fluxo principal O administrador escolhe que tipo de cadastro deseja realizar.
Fluxos de exceção N/A.
Fluxos alternativos N/A.
Pós-condições Uma tela de cadastro de usuário/calendário/exemplar será aberta.

CSU03: Relatório
Sumário O Bibliotecário solicita o relatório desejado.
Atores envolvidos Bibliotecário.
Pré-condições O Bibliotecário deve estar logado.
Fluxo principal O Bibliotecário seleciona que tipo de relatório deseja gerar.
Fluxos de exceção N/A.
Fluxos alternativos N/A.
Pós-condições Uma tela de relatório do mês/exemplar/dia/empréstimo geral será exibida.

CSU04: Empréstimo
Sumário Usuário do sistema solicita empréstimo de um ou mais exemplares.
Atores envolvidos Bibliotecário, Cliente e Leitor RFID.
Pré-condições Usuário deve está cadastrado e logado no sistema.
Fluxo principal • O usuário apresenta o código do exemplar ao leitor de RFID para leitura; • Informações do(s) exemplar(es) são exibidos na tela para a confirmação.
Fluxos de exceção Exemplar não está cadastrado no sistema.
Fluxos alternativos N/A.
Pós condições • O status do livro será alterado para indisponível; • Um comprovante de empréstimo será exibido na tela.

CSU05: Devolução
Sumário Usuário do sistema solicita devolução de um ou mais exemplares.
Atores envolvidos Bibliotecário, Cliente e Leitor RFID.
Pré-condições Usuário deve está cadastrado e logado no sistema.
Fluxo principal • O usuário apresenta o código do exemplar ao leitor de RFID para leitura; • Informações do(s) exemplar(es) são exibidos na tela para a confirmação.
Fluxos de exceção Exemplar não está cadastrado no sistema.
Fluxos alternativos N/A.
Pós-condições O status do livro será alterado para disponível.

CSU06: Posse
Sumário Usuário solicita relatório dos livros em posse
Atores envolvidos Cliente.
Pré-condições Usuário deve estar cadastrado e logado no sistema.
Fluxo principal Uma lista com os exemplares em posse será exibida na tela.
Fluxos de exceção N/A.
Fluxos alternativos N/A.
Pós condições N/A.

CSU07: Cadastro Usuário
Sumário O administrador do sistema realiza o cadastro de algum usuário.
Atores envolvidos Bibliotecário.
Pré-condições Bibliotecário deve está logado no sistema da biblioteca.
Fluxo principal O bibliotecário insere os dados cadastrais do usuário.
Fluxos de exceção • Campo obrigatório em branco.
Fluxos alternativos N/A.
Pós condições Exemplar cadastrado no banco de dados.

CSU08: Cadastro Exemplar
Sumário Bibliotecário insere todos os dados obrigatórios para o cadastro do exemplar.
Atores envolvidos Bibliotecário.
Pré-condições Bibliotecário estar cadastrado e logado no sistema.
Fluxo principal Inserir todos os dados necessários.
Fluxos de exceção Deixar algum campo obrigatório em branco.
Fluxos alternativos N/A.
Pós-condições Exemplar cadastrado no banco de dados.

CSU09: Relatório do Mês
Sumário Gera relatório de total de empréstimos, cadastros e débitos do mês.
Atores envolvidos Bibliotecário.
Pré-condições Bibliotecário deve estar cadastrado e logado no sistema.
Fluxo principal Relatório gerado é exibido na tela
Fluxos de exceção Não haver nenhum "movimento"bibliotecário.
Fluxos alternativos Imprimir relatório gerado.
Pós condições N/A.

CSU10: Relatório do Dia
Sumário Gera relatório de total de empréstimos, cadastros e débitos do dia.
Atores envolvidos Bibliotecário.
Pré-condições Bibliotecário deve estar cadastrado e logado no sistema.
Fluxo principal Relatório gerado é exibido na tela
Fluxos de exceção Não haver nenhum "movimento"bibliotecário.
Fluxos alternativos • Imprimir relatório gerado; • Encaminhar e-mail de aviso ao(s) Cliente(s) em caso de débito.
Pós-condições N/A.

CSU11: Leitura de Código
Sumário Leitor coleta dados da etiqueta RFID.
Atores envolvidos Leitor RFID.
Pré-condições Usuário cadastrado solicitar leitura de código.
Fluxo principal Realizar a leitura do código.
Fluxos de exceção Leitor RFID não realizou a leitura corretamente.
Fluxos alternativos Digitar código de barras da etiqueta do exemplar.
Pós-condições Exemplar adicionado à lista de requerimento de empréstimo.

APÊNDICE B – INSTALAÇÕES DAS FERRAMENTAS UTILIZADAS

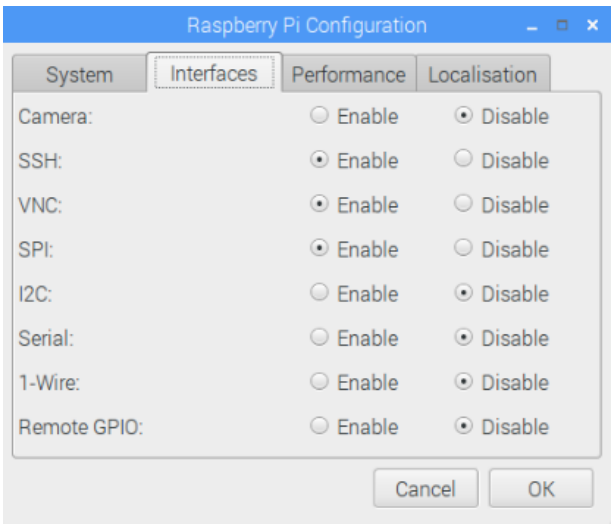
B.1 Raspberry Pi

A aquisição do sistema operacional utilizado no Raspberry foi feita com o NOOBS, que pode ser encontrado no site da organização. Para a instalação do sistema operacional *Raspbian* no Raspberry foi utilizados:

- Raspberry Pi 3 Model B;
- Cartão de memória SD de 8GB;
- NOOBS.

O NOOBS vem compactado, ao descompacta-lo é movido todo seu conteúdo para o cartão de memória. Ao terminar esse processo, o cartão é inserido no Raspberry que é necessário conectar um teclado e/ou um mouse e um monitor por HDMI para auxiliar na instalação correta do sistema.

Figura 23 – Interfaces utilizadas



Fonte – Elaborado pela autora.

Após a instalação as interfaces SSH e VNC foram habilitadas para conexão remota Figura 23, desse modo não precisando de teclado, mouse e monitor conectados ao Raspberry. E interface SPI para a comunicação com o módulo RC522.

B.2 Computador

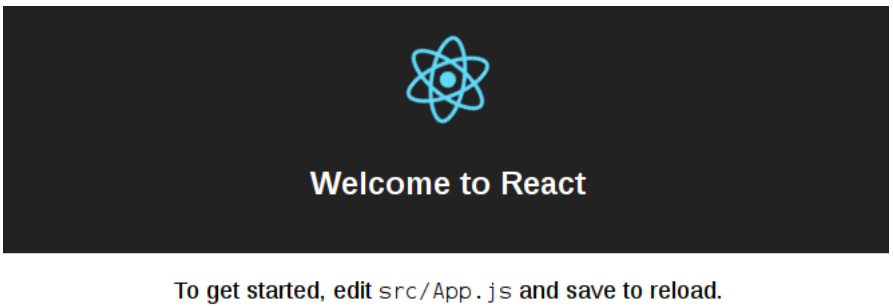
Na máquina que foi utilizada para desenvolver a aplicação, foi necessário instalar o Node.js, que o instalador para cada plataforma pode ser encontrado no site da fundação <https://nodejs.org/en/download/>. Seu gerenciador de pacotes (NPM) o acompanha ao instala-lo.

Para criar uma aplicação ReactJS foi seguidos os seguintes comandos:

1. `npm install -g create-react-app`
2. `create-react-app <nome da aplicação>`

Primeiro comando instala o pacote `create-react-app` para a criação de uma aplicação React e a segunda usa este pacote pra a criação da aplicação. Ao rodar o comando `npm start`, abrirá uma aba no navegador exibindo uma página semelhante à Figura 24.

Figura 24 – Aplicação ReactJS rodando em um navegador.



Fonte – Elaborado pela autora.

A medida do desenvolvimento da aplicação, alguns módulos não nativos do Node.js foram instalados com o comando `npm install <nome do(s) pacote(s)> --save`, salvando às dependências de desenvolvimento no arquivo "`package.json`".

Para a instalação do MongoDB, foi adquirido o instalador também em seu site <https://www.mongodb.com/download-center/community>.