UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO POLITÉCNICO DA UFSM
CURSO DE TÉCNICO EM INFORMÁTICA

Pedro Leal Lovatto

ESTUDO SOBRE O DESENVOLVIMENTO DE APLICAÇÕES WEB COM A LINGUAGEM JAVASCRIPT E A UTILIZAÇÃO DO FRAMEWORK REACT

Santa Maria, RS
2022

Pedro Leal Lovatto

ESTUDO SOBRE O DESENVOLVIMENTO DE APLICAÇÕES WEB COM A LINGUAGEM JAVASCRIPT E A UTILIZAÇÃO DO FRAMEWORK REACT

Trabalho de Conclusão de Curso apresentado ao Curso de Técnico em Informática, Área de Concentração em Linguagens de Programação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Técnico em Informática**.

ORIENTADOR: Prof. Bruno Augusti Mozzaquatro

Santa Maria, RS
2022

Pedro Leal Lovatto

ESTUDO SOBRE O DESENVOLVIMENTO DE APLICAÇÕES WEB COM A LINGUAGEM JAVASCRIPT E A UTILIZAÇÃO DO FRAMEWORK REACT

Trabalho de Conclusão de Curso apresentado ao Curso de Técnico em Informática, Área de Concentração em Linguagens de Programação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Técnico em Informática**.

Aprovado em 27 de abril de 2022:

Bruno Augusti Mozzaquatro, Dr. (UFSM)
(Presidente/Orientador)

Giani Petri, Dr. (UFSM)

Marcos Luís Cassal, Dr. (UFSM)

Santa Maria, RS
2022

RESUMO

ESTUDO SOBRE O DESENVOLVIMENTO DE APLICAÇÕES WEB COM A LINGUAGEM JAVASCRIPT E A UTILIZAÇÃO DO FRAMEWORK REACT

AUTOR: Pedro Leal Lovatto
ORIENTADOR: Bruno Augusti Mozzaquatro

Este trabalho apresenta um estudo sobre a linguagem de programação JavaScript e alguns frameworks dessa linguagem. O objetivo do trabalho é realizar um estudo para se conhecer e aprender o JavaScript, em seguida, foi realizada uma pesquisa e estudo através de vídeos, artigos e outros TCC sobre os frameworks de JavaScript mais populares, o Angular, o Vue.js e o React, dessa maneira um foi escolhido para um estudo mais aprofundado, nesse caso o React. Esse estudo realizado levou para o desenvolvimento de uma aplicação web com foco em demonstrar os recursos da linguagem e do framework. O estudo teórico e prático renderam novos conhecimentos sobre tecnologias de desenvolvimento web emergentes no mercado de trabalho atual, o JavaScript e o React.

Palavras-chave: JavaScript . Framework . React . Aplicação . web .

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais as empresas estão oferecendo seus serviços através da Internet, que traz grande comodidade e praticidade para as pessoas (ANDRADE, 2016). Algo que aumentou de março de 2020 para cá devido a pandemia de COVID-19, segundo uma pesquisa realizada em setembro de 2020 pela empresa Catho, plataforma brasileira com vagas de emprego, a procura por desenvolvedores web aumentou 107% (DIRIGIDA, 2020). A complexidade e exigência dos projetos exigidos em empresas veem aumentando também, por conta disso os desenvolvedores web precisam estudar novas linguagens de programação para atender as demandas de grandes empresas.

Com o aumento de linguagens de programação, notou-se um crescimento na utilização dos frameworks, um framework nada mais é do que um conjunto de classes que proporcionam um ambiente padronizado, deixando o desenvolvimento bem mais produtivo, permitindo o re-uso de código (PAIXãO, 2017).

Existem muitos motivos para o uso de um framework, como os recursos que possuem, oferecendo um aumento da produtividade, segurança, suporte e outros benefícios. Mas uma das principais vantagens é podermos utilizar uma solução pronta e testada para problemas comuns, ou seja, não precisamos reinventar a roda para cada projeto. (WILLIAMMIZUTA, 2017)

1.1 OBJETIVO GERAL

Este trabalho tem como objetivo geral o estudo e aquisição de conhecimentos sobre o JavaScript e o framework React.

1.1.1 Objetivos Específicos

- Realizar um estudo sobre a linguagem de programação JavaScript.
- Realizar uma pesquisa sobre frameworks JavaScript para desenvolvimento web.
- Estudar os recursos disponíveis do framework React.
- Desenvolver uma aplicação web aplicando os recursos do framework React.

Apresentado os detalhes introdutórios do trabalho, a partir do próximo capítulo começará o estudo sobre o JavaScript.

1.2 ESTRUTURA DO TRABALHO

Esse trabalho foi organizado em capítulos. Este capítulo foi uma introdução ao que será aprofundado nos próximos capítulos, as motivações para o trabalho e os objetivos.

O capítulo 2 apresenta os conceitos básicos do JavaScript, falando como a linguagem funciona, seus tipos de dados e funções.

No capítulo 3 está o estudo sobre os frameworks Angular, Vue.js e React, além de uma tabela que compara os três frameworks.

O capítulo 4 apresenta um estudo mais aprofundado do React, detalhando seus recursos e motivos para esse framework ter sido escolhido para o trabalho.

No capítulo 5 está a apresentação da aplicação que foi desenvolvida utilizando o framework React.

O capítulo 6 concluí o trabalho, apontando benefícios adquiridos com o estudo e dificuldades encontradas.

2 LINGUAGEM JAVASCRIPT

2.1 CONCEITOS BÁSICOS

A linguagem JavaScript é destinada para back-end [1] e front-end [2] e foi criada em 1996 por Brendan Eich, o qual na época era um desenvolvedor na Netscape. A linguagem JavaScript foi desenvolvida para rodar no lado do cliente, ou seja, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário. Isso é possível porque existe um interpretador JavaScript hospedado no navegador (SILVA, 2010).

A linguagem JavaScript se organiza em sentenças de código, blocos de código (agrupamento de sentenças de código). Para executar a linguagem JavaScript existe várias maneiras, como ferramentas online, console do navegador, o software Node ou um editor, como *Visual Studio Code*. O desenvolvimento de sistemas web com a linguagem Javascript faz uso de três palavras reservadas: `var`, `let` e `const`.

O uso da palavra reserva `var` tem a característica de utilização na função atual ou também no escopo global, apesar de não recomendado para "redeclarar"uma variável. Por outro lado, a palavra reservada `let` é utilizada para declarar variáveis em um escopo de bloco, instrução ou em uma expressão na qual está sendo utilizada.

Com base nisso, o `var` possui algumas limitações relacionadas ao escopo. Já o `let` é mais utilizado devido a flexibilidade do seu uso, pois em certas ocasiões o usuário tem a possibilidade de declarar uma variável para apenas dentro do escopo de uma função, e não globalmente. Essa é a principal diferença entre a palavra reservada `let` e a `var`.

A linguagem JavaScript possui também uma palavra reservada para definição de constantes. O `const` é utilizado para definir um valor que não irá mudar durante a execução do código.

2.2 TIPOS DE DADOS

Nesta seção serão descritos os principais tipos de dados da linguagem JavaScript, tais como number, string, boolean, *array*, *object*/objeto, *null* e *undefined*. (ZAKAS, 2012)

O tipo de dado *number* oferece recursos para trabalhar com valores numéricos.

[1] Back-end é a estrutura de código que roda do lado do servidor para o processamento das regras de negócio da aplicação.

[2] Front-end é a estrutura de código responsável pela definição da interface gráfica da aplicação que o usuário deverá interagir.

Esse tipo permite armazenar dados numéricos como *int* e *float* e também a função *Number*, conforme o Código 2.1.

```
1 const peso1 = 1.0
2 const peso2 = Number('2.0')
```

Código 2.1: Exemplo de utilização de tipo de dados *number*

Outro tipo de dado é o *String* que é destinado para armazenar uma cadeia de caracteres no JavaScript. O Código 2.2 apresenta um exemplo de utilização do tipo de dado String.

```
1 const escola = "Cod3r"
```

Código 2.2: Exemplo de uso do tipo de dado *String*.

O tipo de dado *String* disponibiliza alguns métodos para manipulação e acesso as informações como pode ser visualizado no Código 2.3. A linha 1 apresenta o método *charAt()* para retornar o caractere localizado no índice informado no parâmetro da função. Já o método *indexOf()* retorna o índice que contém o caractere informado no parâmetro da função.

```
1 console.log(escola.charAt(4))
2 console.log(escola.indexOf('3'))
```

Código 2.3: Exemplos de utilização dos métodos do tipo de dados *String*.

Em algumas linguagens, como a linguagem Java, o tipo de dado *boolean* é considerado verdadeiro (*true*) ou falso (*false*). Já em outras linguagens são aceitos os valores 0 (falso) e 1 (verdadeiro). Para o JavaScript, além de aceitar verdadeiro e falso e também 0 e 1, também aceita pontos de exclamação como verdadeiro ou falso, dependendo da maneira de como são usados. O Código 2.4 apresenta alguns exemplo de uso do tipo de dado boolean.

```
1 let isAtivo = true
2 console.log(!isAtivo)
3 console.log(!!isAtivo)
```

Código 2.4: Exemplos de utilização do tipo de dado *boolean*.

A linha 1 apresenta apresenta a atribuição de verdadeiro (*true*) para a variável *isAtivo* e na linha 2 é mostrada o valor da variável passando apenas um símbolo de exclamação antes do nome da variável. Dessa forma, o valor será o inverso do valor armazenado na variável que é falso (*false*). Por outro lado, a linha 3 apresenta o valor com a passagem de dois símbolos de exclamação antes do nome da variável. Isso significa que um símbolo anula o outro e o resultado será verdadeiro (*true*).

Na sequência, outro tipo de dado utilizado na linguagem JavaScript é o *Array*. Esse tipo de dado é uma forma de agrupar múltiplos valores de uma forma linear, como se fosse

uma lista de elementos. O acesso aos elementos do *array* é através de índices, iniciando pelo índice 0.

O tipo de dado *Array* é uma estrutura de dados destinada a armazenamento de dados homogêneos, onde não é recomendado o armazenamento de diferentes tipos de dados no mesmo *array*.

O tipo de dados *Object* em JavaScript é uma coleção de chaves e valores onde tem o nome do atributor e o valor correspondente. Esse valor pode ser um número, texto, *boolean* e até uma função. Além disso, é possível armazenar objetos dentro de um objeto. JavaScript permite, além de criar um objeto, definir como esse objeto é, quais são seus atributos, comportamentos e funções dinamicamente. O Código 2.5 apresenta um exemplo de criação de um *object* com a definição de um atributo nome.

```
1 const prod1 = {}
2 prod1.nome = "Celular Ultra Mega"
```

Código 2.5: Exemplo da criação de um *Object*.

A linguagem JavaScript possui o conceito de *undefined* e consiste que uma variável foi declarada, mas não inicializada, ou seja, nenhum valor foi atribuído. Por outro lado temos o conceito de *null*. O *null* significa que o usuário inicializou uma variável e que ela não possui atribuição de valor, ou seja, não aponta para nenhum local de memória. Portanto, alguns novos usuários confundem *underfined* e *null*, e esses conceitos necessitam estar bastante esclarecidos para evitar potenciais problemas durante o desenvolvimento de aplicações.

2.3 ESTRUTURAS DE CONTROLE

A linguagem JavaScript, assim como outras linguagens de programação também dispõe várias estruturas de controles que permitem ao usuário estabelecer condições ou gerenciamento em determinadas etapas do código. Nesta seção iremos descrever as principais estruturas de controles disponíveis pela linguagem JavaScript, tais como estrutura condicionais e laços de repetição. (ZAKAS, 2012)

A estrutura condicional *if* é usada em momentos que é desejado mostrar só resultados em uma condição específica (Código 2.6).

```
1 NoticiaBoa(8.1)
2 NoticiaBoa(6.1)
3 function NoticiaBoa(nota) {
4   if(nota >= 7){
5       console.log("Aprovado com " + nota)
6   }
```

```
7 }
```

Código 2.6: Exemplo de *if*.

A estrutura condicional *if/else* é uma alternativa/continuação para o *if*, sua utilidade seria para quando o usuário quer comparar valores, ou se é verdadeiro ou falso (Código 2.7).

```
1 function NoticiaBoa(nota) {
2   if(nota >= 7){
3         console.log("Aprovado")
4     }
5     else{
6         console.log("Reprovado")
7     }
8  }
```

Código 2.7: Exemplo de *if/else*.

A estrutura condicional *switch* é utilizada para especificar múltiplas seleções (Exemplo no Código 2.8).

```
1 const imprimirResultado = function (nota){
2     switch(Math.floor(nota)){
3         case 10:
4         case 9:
5         case 8:
6       case 7:
7           console.log("Aprovado")
8             break
9         case 6:
10        case 5:
11        case 4:
12            console.log("Reprovado")
13            break
14        default:
15            console.log("Nota inválida")
16    }
17 }
```

Código 2.8: Exemplo de *switch*.

A estrutura de controle para laço de repetição é utilizado quando um bloco de código necessita ser executado múltiplas vezes até que o limite seja alcançado. O laço de repetição *while* é usado bastante para situações em que o usuário tem um número indeterminado de repetições, mas pode ser usado para um número determinado também. O Código 2.9 apresenta um exemplo do código com laço de repetição *while*.

```
1 let contador = 1
2 while(contador <= 10){
```

```
3       console.log(`contador = ${contador}`)
4       contador++
5   }
```

Código 2.9: Exemplo de *while*.

O laço de repetição *do/while* inverte um pouco a lógica do laço de repetição *while*, garantindo que pelo menos uma vez haja repetição no laço. O laço de repetição *do/while* executa a primeira repetição, e só depois ele irá fazer o teste condicional (Código 2.10).

```
1   do{
2       console.log(`contador = ${contador}`)
3       contador++
4   } while(contador <= 10)
```

Código 2.10: Exemplo de *do/while*.

O laço de repetição *for* é uma estrutura de controle usada para casos que o usuário deseja repetir um bloco de código até que uma condição seja válida. O Código 2.11 apresenta um exemplo de código que é apresentado o valor de 1 a 10.

```
1   let contador = 1
2   for(let i = 1;i <= 10; i++){
3       console.log("i = "+i)
4   }
```

Código 2.11: Exemplo de for.

A estrutura de controle *break/continue* seriam duas expressões usadas em laços de repetição, break para interromper e continue para ignorar.

2.4 FUNÇÕES DO JAVASCRIPT

Nessa próxima sessão, será detalhado algumas das principais funções existentes no Javascript. Uma função em Javascript seria um dado, um dado especial, porque ele é executado e é possível também passar através dos parênteses os parâmetros para executar. Uma função é uma das partes mais importantes da linguagem Javascript.

O primeiro tipo de função é a *função literal* e essa função pode receber parâmetros e retornar um valor. O retorno de um valor de uma função no Javascript é opcional. Caso a função não retorne nada, então, ela retornará o tipo de dado *undefined*. Um detalhe, mesmo quando há uma função de uma única sentença, não é possível omitir o bloco, pois ele é obrigatório (CROCKFORD, 2008). O Código 2.12 apresenta um exemplo de utilização de uma função literal.

```
1   1 function fun1 () { }
```

Código 2.12: Exemplo de função literal

Além de poder criar funções de maneira literal, é possível armazenar uma função em uma variável, dessa maneira o usuário será capaz de referenciar a função a partir dessa variável.

```
1 const fun2 = function () { }
```

Código 2.13: Outro exemplo de função literal dessa vez armazenando-a em uma variável.

```
1 const array = [function (a, b) { return a + b }, fun1, fun2]
```

Código 2.14: Função dentro de um array

Um aspecto importante para funções na linguagem Javascript seriam os *parâmetros*, pois são opcionais. No entanto, também é possível passar quantos o usuário necessitar, embora a função irá limitar os parâmetros que foram declarados e irá ignorar os demais. As funções sem parâmetros não significam que não é possível passar nenhum parâmetro, mas esse recurso oferece facilidade na transmissão de dados entre funções da linguagem.

```
1  function soma (){
2      let soma = 0
3      for(i in arguments){
4          soma += arguments[i]
5      }
6      return soma
7  }
8  console.log(soma)
9  console.log(soma(1))
10 console.log(soma(1.1, 2.2, 3.3))
```

Código 2.15: Exemplo de parâmetros.

Note o '*arguments*', ele é um array interno, toda função tem esse array disponível. Quando nenhum parâmetro é passado, esse array é vazio e o usuário tem a possibilidade de pegar todos os parâmetros que foram passados a partir da chamada de uma função e fazer qualquer tipo de operação. Nesse exemplo, pegar cada parâmetro e somar. (CROCKFORD, 2008)

Não será detalhado todos os tipos de parâmetros, então, serão mostrados apenas um tipo como exemplo. O *Parâmetro padrão* já existia na versão anterior a de 2015, mas era bem mais complicada e que podia gerar potenciais problemas. A versão de 2015 trouxe essa possibilidade de ter o parâmetro padrão de uma maneira melhor, pois antes era necessário fazer verificação dos valores. A partir da versão de 2015 não precisa mais.

```
1 function soma3(a = 1, b = 1, c = 1){
2     return a + b + c
3 }
4 console.log(soma3())
5 console.log(soma3(3))
```

```
6 console.log(soma3(1, 2, 3))
```

Código 2.16: Exemplo de parâmetro padrão.

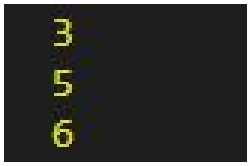

Figura 2.1 – Resultado do Código 2.16.

Um tipo de função bem comum de ser encontrada em aplicações que utilizam Javascript são as *funções Arrow*, elas foram introduzida na versão 2015, tem como objetivo possuir uma sintaxe reduzida e ter um `this` associado ao contexto no qual a função foi escrita. (BANKS; PORCELLO, 2017)

```
1 // Sem função arrow
2 let dobro = function(a) {
3     return 2 * a
4 }
5 // Com função arrow
6 dobro = a => 2 * a
```

Código 2.17: Exemplo de função arrow.

Como é possível notar, as funções *arrow* reduzem bastante a sintaxe. Em alguns casos, o usuário tem apenas um parâmetro (Código 2.17), é até possível escrever tudo em uma única linha, algo que é bem usado em funções fazem apenas um trabalho. Isso favorece o reuso, pois o usuário está fazendo uma única coisa, ajudando a ter um código um pouco mais funcional e é útil ter funções com apenas uma linha de código, sem necessariamente ter que escrever todo o bloco.

A palavra-chave *this* em funções *arrow* é fixa, ou seja, é baseado no contexto que a função foi escrita e não é influenciado pela função ser chamada de locais diferentes.

Um detalhe é que uma função *arrow* sempre é uma função anônima. Funções anônimas seriam funções sem nome, podendo serem usadas para passar as funções como parâmetro como se fosse um objeto qualquer, ou seja, se o usuário quer chamar a função *Arrow* depois, terá que armazenar ela em alguma variável ou constante.

Outro tipo de função são as funções *Callback*. Essas funções têm como objetivo passar uma função para outra função como um parâmetro, e após um determinado evento acontecer, essa função que foi passada será chamada de volta. Funções *Callback* podem ser chamadas várias vezes e são mais utilizadas em funções assíncronas, em que uma função precisa esperar por outra função. (CROCKFORD, 2008)

```
1 const fabricantes = ["Mercedes", "Audi", "BMW"]
2 function imprimir(nome, indice){
```

```
3       console.log(`${indice + 1}. ${nome}`)
4   }
5   fabricantes.forEach(imprimir)
```

Código 2.18: Exemplo de função Callback.

A função *ForEach* é destinada a utilização com tipos de dados *Array* e executa uma dada função em cada elemento do *Array*. Dessa forma, esse tipo de função executa funções do tipo *Callback* em cada elemento da ordem com um valor atribuído considerando três argumentos: o valor do elemento, índice do elemento e o *array* que está sendo percorrido. Então, essa chamada consiste na chamada da função imprimir onde o forEach vai passar como parâmetro o nome do elemento e como segundo parâmetro o índice.

Para finalizar essa seção, existem as funções *Factory*, que seriam funções que retornam novos objetos, porém não necessitam da palavra-chave `new`. Esse recurso da linguagem JavaScript oferece simplicidade na criação de instâncias de objetos minimizando a complexidade da criação de classes com a palavra-chave *new* (ELLIOTT, 2017).

```
1   function criarPessoa(nome, sobrenome){
2       return{
3           nome: nome,
4           sobrenome: sobrenome,
5           nomeCompleto() {
6               nome + " " + sobrenome;
7           }
8       };
9   }
10  let pessoa1 = criarPessoa("Pedro", "Lovatto");
11  let pessoa2 = criarPessoa("Márcio", "Luís");
12  console.log(criarPessoa())
```

Código 2.19: Exemplo de função Factory.

O Código 2.19 apresenta o uso da chamada da função *criarPessoa*, onde são criadas instâncias de um objeto, ou seja, criando novos objetos a partir da chamada da função. Dessa forma, basta o usuário passar as informações como parâmetros, sem a necessidade de escrever uma nova função toda vez que o usuário desejar criar uma pessoa com informações diferentes (Linha 10 e 11). Isto é conhecido como Programação orientada à objetos, que nada mais é do que um modelo de programação onde diversas classes possuem características que definem um objeto na vida real. Cada classe determina o comportamento do objeto definido por métodos e seus estados possíveis definidos por atributos.

O capítulo apresentou uma introdução ao JavaScript, mostrando seus conceitos básicos, particularidades, seus tipos de dados e funções. Como pode ser visto durante o capítulo, existem vários tipos de funções no JavaScript, cada uma podendo ser utilizada em uma determinada situação. O próximo capítulo irá apresentar de maneira mais detalhada

os frameworks do JavaScript e o estudo que foi realizado sobre cada um.

3 FRAMEWORKS JAVASCRIPT

Os três principais frameworks front-end, Angular (ANGULAR, 2016), React (REACT, 2013) e Vue.js (VUE.JS, 2014), se destacam pelo modo de desenvolvimento baseado no modelo *Single Page Application* [3], mas é possível trabalhar com modelos convencionais também. Em modelos convencionais, toda vez que o usuário navega pela aplicação é necessário recarregar todo o conteúdo da página, até partes estáticas, isso dificulta bastante quando a aplicação era acessada por dispositivos móveis, pois esses dispositivos tendem a ter conexões 4G ou 3G, que são conexões lentas, aumentando o problema.

Aplicações SPA corrigem esses problemas por meio da ideia que toda vez que será exibido na tela uma nova informação não é necessário recarregar todo o conteúdo da página, ao invés disso, será atualizado somente a parte da tela que essa informação será exibida, por exemplo, caso o usuário clique em uma informação de um menu lateral, não é necessário recarregar esse cabeçalho (e nem o menu), apenas atualizar a porção da tela onde esse formulário ou outra opção será exibida.

Algo que é importante destacar é a diferença entre um Framework e uma Biblioteca. Bibliotecas seriam um conjunto de códigos em que o usuário pode usar apenas o que o interessa, então uma Biblioteca pode reunir um conjunto de funções, classes..etc, mas o usuário só irá utilizar a parte que ele precisa.

Já um Framework seria um conjunto de Bibliotecas, um esqueleto que o usuário utiliza para criar seu código em cima dele, mas é necessário que o usuário siga o padrão do framework, ou seja, as estruturas já pré-definidas. Então, em resumo, Biblioteca é o que o usuário usa no código e o framework usa o código. Frameworks acabam se destacando por sua padronização, redução de custos, facilidade de manutenção, entre outros detalhes.

3.1 ANGULAR

Lançado pelo Google em 2010, Angular evoluiu muito desde então, em 2016 recebeu uma nova versão, chamada Angular 2, atualmente o Angular está na versão 10. Angular é construído em cima do TypeScript, que seria uma ferramenta que permite programar Javascript de forma tipada. O Angular é uma das principais escolhas de empresas grandes para projetos front-end devido a sua robustez e padronização.

Antes do Angular 2, existia o AngularJS, mas a versão 2 já está muito à frente, por ter uma performance melhor, suporte a mobile e suporte do Google, AngularJS não possui suporte a mobile além de não ter novas atualizações, a não ser para corrigir erros. A seguir

[3] SPA consiste em aplicações que carregam dinamicamente o conteúdo sem necessidade de atualização da página (SCHNEIDER, 2016)

serão apresentados os diretórios de um ambiente de trabalho de um projeto Angular:

- **e2e:** contém configurações específicas para testes unitários.
- **node modules:** contém todos os pacotes necessários para trabalhar em Angular, é possível instalar outros pacotes.
- **src:** seria o diretório principal, contém arquivos como main.ts, polyfills.ts, index.html, styles.css e test.js. O que vale destacar é o polyfills.ts, que seria como um tradutor, pois na hora de transpilar (ou seja, converter um código fonte em outro por meio de um processo de compilação) ele traduz para o navegador, fazendo com que o usuário tenha uma melhor visualização em qualquer tipo de navegador. Dentro do diretório src contém a segunda pasta principal, o app, que contém todos os componentes da aplicação, cada componente terá três arquivos, um CSS, um html e outro spec. Além disso, este diretório contém *assets*, que irá conter todos os arquivos externos que é possível colocar na aplicação, como imagens.

  Outro diretório no *src* é o *environments*, que é um diretório que contém a configuração de se a aplicação vai estar gerando um *build*.
- **angular.json:** contém todas as configurações do Angular, como qual é o favicon, qual seria o styles.css geral.
- **package.json:** lista todos os pacotes que tem dentro da aplicação.
- **tsconfig.json:** arquivo que mostra qual versão do ECMAScript está sendo usado.

A Figura 3.1 apresenta a estrutura de diretórios do framework Angular.

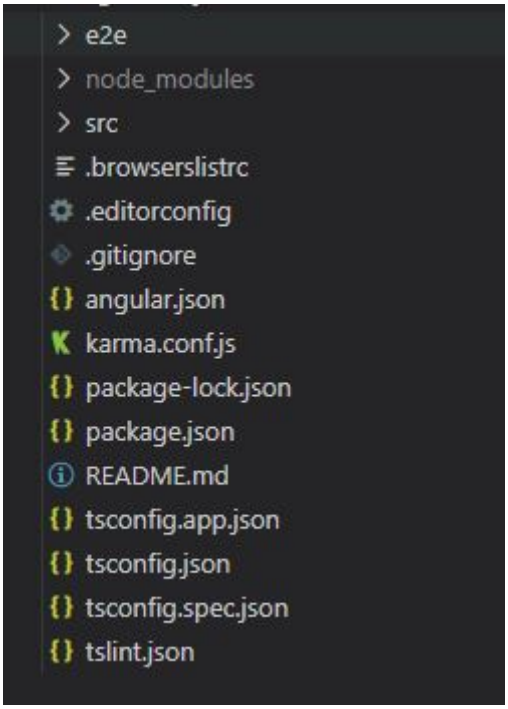

Figura 3.1 – Hierarquia de diretórios do framework Angular.

3.2 VUE.JS

Vue.js é um dos frameworks mais recentes, foi desenvolvido e lançado em 2014 por Evan You, o Vue.js vem se tornando cada vez mais popular. Diferente do Angular e do React, o Vue.js não possui suporte de uma grande empresa, e sim pela comunidade por meio do Patreon. O Vue.js se destaca por ser reativo, ou seja, a sua arquitetura atualiza a camada visual toda vez que tem uma mudança de estado, então o HTML será atualizado toda vez que o valor de uma variável de um componente no framework for alterado. Outra característica do Vue.js é ele ser progressivo, o que quer dizer que ele é adaptável com outros frameworks e bibliotecas, então é possível usar o Vue.js em uma parte de uma aplicação já construída ou trabalhar em conjunto, na parte visual, com framework back-end.

Por ter surgido depois do React, o Vue.js usa algumas das suas características, como o *VirtualDOM*, mas apesar disso, o Vue.js trabalha de maneira diferente, além de ser mais completo. A Figura 3.2 apresenta a estrutura de diretórios do framework.

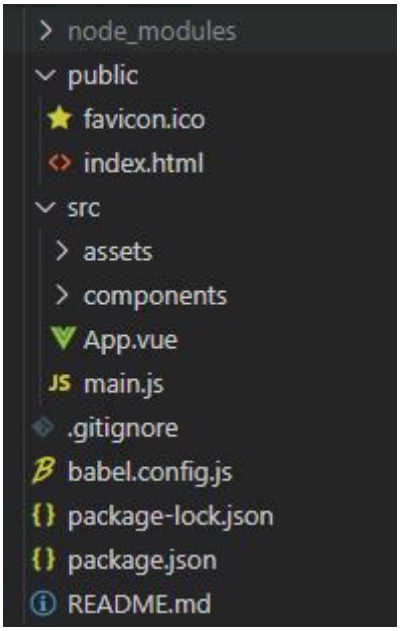

Figura 3.2 – Hierarquia de diretórios do Vue.js.

- **README:** mostra alguns comandos, como executar o servidor.
- **package.json e package-lock.json:** arquivos que o Node e o npm usam para gerenciar quais bibliotecas o projeto usa.
- **babel.config.js:** contém as configurações do Babel, transpilador que converte ECMAScript 6 para ECMAScript 5, pois é a versão mais compreensível para o navegador.
- **node modules:** contém as bibliotecas que o projeto usa.
- **public:** contém dois arquivos, favicon.icon (ícone que aparece na aba) e index.html (ponto de entrada para sua aplicação, onde o vue será inserido.
- **src:** parte em que o usuário irá programar, contém duas pastas e dois arquivos: *assets* (contém as imagens do projeto), *components* (diretório que o Vue.JS recomenda colocar os seus componentes).
- **App.vue** (componente raíz da aplicação, ele está inserido no arquivo index.html de public).
- **main.js** (responsável por criar o projeto Vue, mostrar as configurações do projeto, mostrar qual será o componente raíz/principal e renderizar a aplicação no id informado dentro do arquivo html).

3.3 REACT

React seria uma biblioteca com foco somente em *UI (user interface/interface de usuário)*, lançada em 2013 pelo Facebook, foi usado em projetos como o próprio Facebook, Instagram, Netflix e Walmart.

No React, o usuário cria componentes, componentes esses que recebem propriedades. Uma vantagem do React é a sua agilidade, devido a sua tecnologia chamada de *JSX*, também chamado de DOM virtual, a importância desse recurso deve por ele renderizar apenas as mudanças que acontecem no DOM, ou seja, ele compara o seu DOM atual e o DOM com as mudanças e mostra para você apenas o resultado.

Não há uma manipulação direta no DOM, isso deixaria a aplicação mais lenta. É recomendado o Babel, interpretador que faz com que o código Javascript seja interpretado pela maioria dos navegadores.

A Figura 3.3 apresenta a estrutura de diretório do ambiente de desenvolvimento para projetos com o framework React.

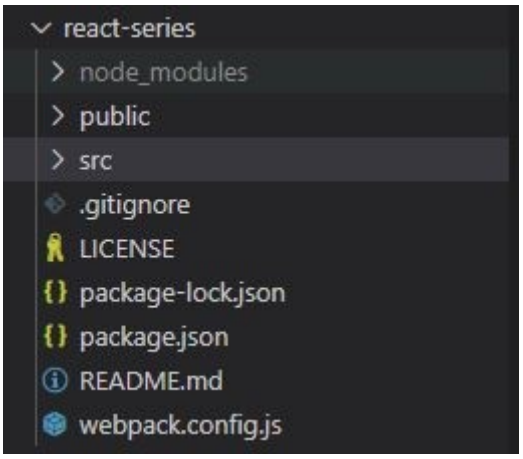

Figura 3.3 – Hierarquia de diretórios do React.

- **node modules:** contém instalações das dependências.
- **public:** contém os arquivos públicos.
- **src:** contém os arquivos privados, onde será criado os scripts e componentes.
- **package.json:** onde é declarado as dependências, como babel, react e react-dom.
- **webpack.config.js:** contém as configurações do webpack.

3.4 COMPARAÇÃO ENTRE OS FRAMEWORKS:

A Tabela 3.1 apresenta os critérios de avaliação considerados neste estudo para verificar aspectos relevantes no desenvolvimento de aplicações SPA.

ITEM	ANGULAR	VUE.JS	REACT
Conhecimento prévio	É necessário um conhecimento em Node.js, Typescript e Webpack.	É necessário apenas um conhecimento básico em HTML e Javascript.	É necessário um conhecimento em Node.js, Typescript, além de familiarizar-se com JSX.
Flexibilidade	Recomendado para projetos grandes.	Recomendado para projetos pequenos.	Recomendado tanto para projetos pequenos quanto para projetos maiores.
Complexidade	Alta.	Muito baixa	Depende do conhecimento prévio do usuário.
Performance	Pesada e tende a ser lenta.	Leve e rápida.	Leve e rápida
Documentação	Ótima documentação.	Satisfatória.	Muito limitada.
Peso das bibliotecas após o build	Depende do *bundle* criado, mas no geral tende a ser maior que o Vue.js e React.	91 KB	116 KB

Tabela 3.1 – Comparativo entre os frameworks Javascript.

É possível identificar que há uma grande diferença nas tecnologias exigidas para iniciar os estudos para cada framework Javascript. Por outro lado, cada framework possui uma característica para utilização em diferentes níveis de projetos no qual o Angular é destinado para grandes projetos, como SPA monolítico, já o Vue.JS é recomendado para projetos pequenos e o React é bastante flexível nesse ponto de vista, sendo utilizado em projetos grandes e pequenos (WOHLGETHAN, 2018).

Quanto à complexidade, o Angular seria o mais complexo entre os três frameworks, pois exige conhecimentos prévios em outras linguagens, tais como Node.js, Webpack e especialmente Typescript, causando problemas para usuários que vieram apenas do Javascript. O React também exige conhecimentos prévios, mas o mais necessário é a familiarização com o *JSX*. O Vue.js é o mais simples de todos, o máximo que o usuário precisa conhecer é o básico de HTML e Javascript. (WOHLGETHAN, 2018).

Em performance, o Angular possui a performance mais lenta, pois utiliza o DOM real, o que força a página da web ser renderizada novamente em qualquer alteração, mesmo se a alteração for pequena. O React e Vue.js sobressaem nesse quesito, pois ambos utilizam o DOM virtual, que seria uma cópia do DOM real, quando as alterações são feitas no DOM virtual, apenas as componentes modificados são renderizados no DOM real. (WOHLGETHAN, 2018).

A documentação é um ponto importante, pois ajuda os desenvolvedores em casos de dúvidas, nisso o Angular se destaca, sua documentação é muito detalhada, fornecendo todas as informações necessárias. O Vue.js não possui uma documentação muito detalhada como a do Angular, mas é ainda boa o bastante para ajudar os desenvolvedores, quanto ao React, possui a pior documentação entre os três, já que nem sequer possui uma documentação oficial(SAKS, 2019).

Em componentes, o Angular possui uma separação clara nos componentes, deixando a interface em arquivos CSS e HTML e comportamento do componente em um código Javascript. Já no Vue.js e o React a interface e comportamento do componente estão em um só lugar, então a mesma parte do código é responsável por criar um elemento de interface e editar seu comportamento(SAKS, 2019).

Este capítulo apresentou os três principais frameworks, Angular, Vue.js e React, mostrando detalhes de cada um, além de uma comparação entre os três. O próximo capítulo irá iniciar o estudo mais aprofundado do React.

4 FRAMEWORK REACT

Após a apresentação e comparação entre os frameworks, o presente trabalho apresentará detalhes sobre o Framework React para focar e aprofundar mais sobre, o motivo seria seus grandes pontos positivos, como sua flexibilidade que permite criar grandes ou pequenos projetos, e também, por utilizar recursos como VirtualDOM.

Apesar do React ter seus pontos negativos, é um Framework que continua em alta desde o seu lançamento, principalmente no mercado de trabalho, pois é usada por grandes empresas, como Globo.com, Walmart, Netflix...etc. O React também está crescendo em popularidade, segundo uma pesquisa do site Statista em que foi pergutado qual framework o usuário utilizava, 40% dos entrevistados relataram usar o React.

A Figura 4.1 apresenta as estatísticas dos frameworks mais utilizados em 2021.

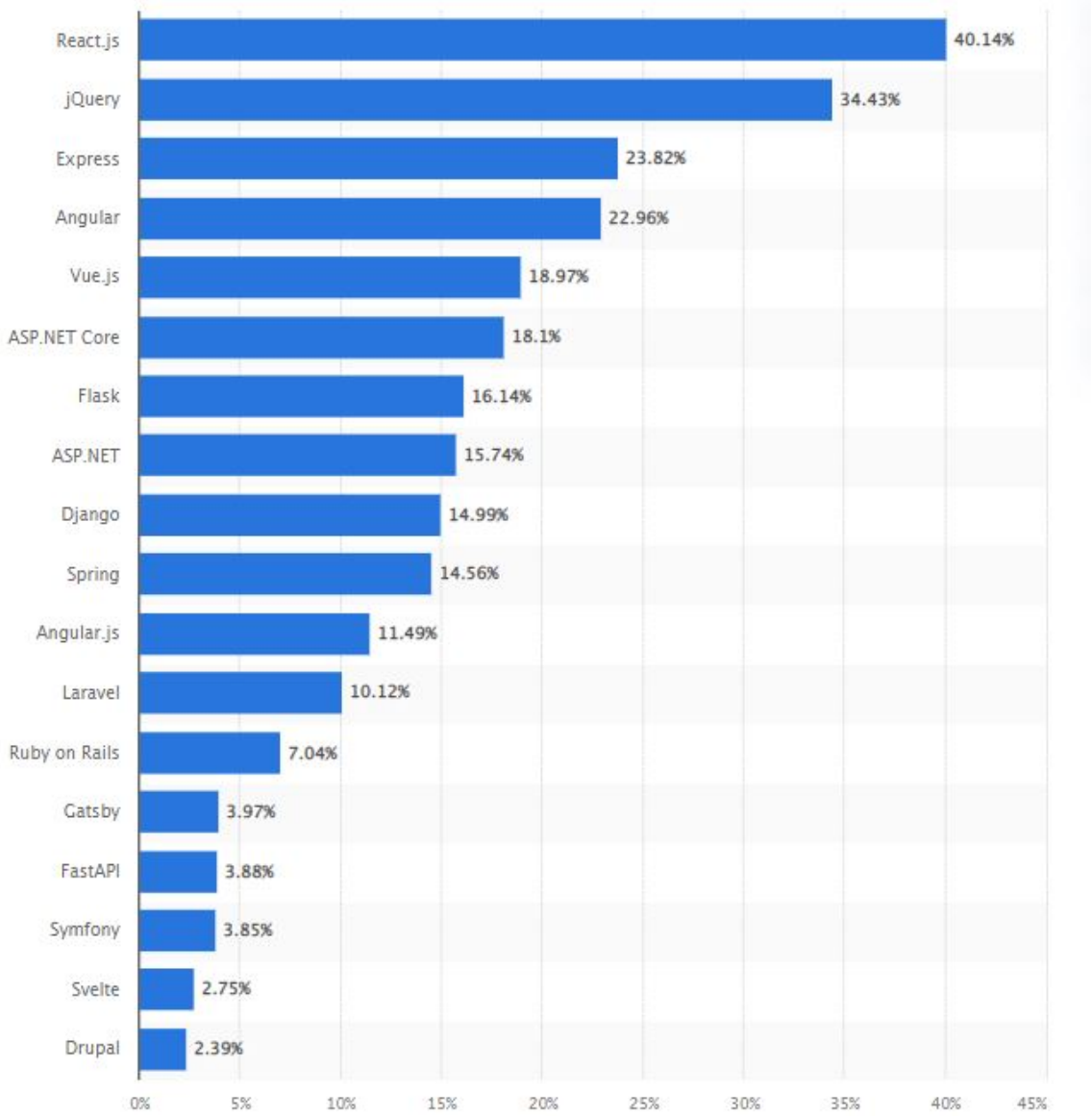

Figura 4.1 – Estatísticas dos frameworks mais utilizados em 2021 (STATISTA, 2022).

Nesta seção serão apresentados os detalhes sobre os recursos disponíveis no framework React.

4.1 SINTAXE JSX

JavaScript XML (JSX) é uma sintaxe de extensão para a linguagem JavaScript escrita em estilo XML. O JSX não é obrigatório, mas fornece uma maneira sintática para substituir métodos como *React.createElement ()*. O código escrito em JSX será convertido em JavaScript para que o navegador possa entender o código. (LUONG, 2019)

```
1 const elemento = <h1>Com JSX!</h1>;
```

Código 4.1: Com JSX

```
1 const elemento = React.createElement('h1', {}, 'Sem JSX');
```

Código 4.2: Sem JSX

Perceba que no Código 3.2 o usuário irá sempre precisar chamar a função *React.createElement* e passar a tag HTML ('h1'), alguma propriedade e algum texto que será renderizado na tela ('Sem JSX'). Já no Código 3.1 bastou usar uma tag simples de HTML, simplificando muito mais para a escrita do código.

4.2 ESTADO DE UM COMPONENTE (*STATE)*

O state de um componente React, representa o estado em que o mesmo se encontra e pode ser composto por diversos atributos. Os atributos que representam um estado são previamente definidos no React e ao receberem novos valores resultam em uma atualização, uma renderização nova do componente que tende a trazer consigo novos resultados visíveis na página web, como uma troca de cor e troca de texto (MOREIRA et al., 2020).

```
1  class Pessoa extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        Nome: "Pedro",
6        Sobrenome: "Lovatto",
7        Idade: 23
8      };
9    }
10   mudarNome = () => {
11     this.setState({Nome: "Márcio"});
```

```
12    }
13    render() {
14      return (
15        <div>
16          <h1>Meu nome é:  {this.state.Nome}</h1>
17          <button
18            type="button"
19            onClick={this.mudarNome}
20          >Mudar o nome</button>
21        </div>
22      );
23    }
24  }
```

Código 4.3: Exemplo de State.

O State está estabelecido na Linha 4 e suas informações estabelecidas nas 5, 6 e 7. Para referenciar um objeto do state é necessário utilizar a sintaxe this.state.(objeto), assim como pode ser visualizado na linha 16 do Código 3.3. Para se alterar um valor do state o usuário deve usar o setState, como visto na linha 11, no caso do Código 3.3 foi estabelecido um evento chamado "mudarNome"dentro do botão de "mudar o nome", dessa maneira, quando o usuário clicar nesse botão a função "mudarNome"será chamada e irá alterar o valor "Pedro"para "Márcio"no state.

4.3 COMPONENTES

Os componentes são bits de código independentes e reutilizáveis. Eles têm a mesma finalidade que as funções JavaScript, mas funcionam isoladamente e retornam HTML por meio de uma função render () (W3SCHOOLS, 2022).

Componentes permitem ao usuário a dividir a interface em várias partes e pensar em cada de maneira isolada.

O desenvolvimento baseado em componentes agora é uma maneira muito popular de construir interfaces de usuário (UI) e aplicativos da web. Quando os programadores desenvolvem o projeto React, os componentes ajudam a economizar muito tempo porque esses programadores não precisam construir as pequenas partes da IU, como botão, campo de texto ou barra superior do projeto. Os desenvolvedores apenas usam os componentes e combinam os componentes para desenvolver o layout (PHAN, 2020).

4.4 CLASSIFICAÇÃO DE COMPONENTES

Os componentes podem ser escritos em forma de funções ou de classes. Os componentes de funções recebem propriedades (props) e retornam um JSX que tem um código *HTML* com as propriedades que o usuário deseja mostrar na tela. Esse recurso serve para uma tarefa específica e são destinados para a apresentação.

```
1 function BemVindo(props) {
2     return <p>Olá, {props.nome}</p>;
3 }
```

Código 4.4: Componente de função

Os componentes de classe extende de "Component", possui um construtor e um método render que irá retornar o JSX. Eles são multitarefas, podem trabalhar com propriedades, fazem gerenciamento de estado e utilizam métodos de ciclo de vida. Possuem a lógica principal da aplicação, fazem geralmente a orquestração entre diversos componentes.

```
1 class BemVindo extends Component {
2   constructor(props) {
3     super(props)
4     this.state = {}
5   }
6   render() {
7     return <h1>Olá, {this.props.nome}</h1>;
8   }
9 }
```

Código 4.5: Componente de classe

4.5 HOOKS

Introduzidos na versão 16.8 do React (REACT, 2022), hooks são um tipo especial de função que possibilita o usuário criar um valor reativo, o que permite o usuário alterar esse valor quando quiser. Hooks não funcionam dentro de classes, eles permitem que o usuário use React sem classes. Hooks são reutilizáveis e podem ser criados pelo próprio usuário, apesar do React já disponibilizar alguns.

```
1 import { useState } from "react";
2 const AdicionarUsuario = () => {
3     const [nome, setNome] = useState('');
```

Código 4.6: Exemplo de um Hook.

O array seria para pegar dois valores que esse hook retorna, o primeiro valor seria o nome, que equivale ao ' ' e o segundo seria uma função para alterar esse valor. O valor dentro do useState seria o valor inicial e pode ser de qualquer tipo, um array, object, boolean, number.

4.6 PROPS

Props permitem transferir dados de um componente para outro. O componente que contém os dados é chamado de componente pai e o componente que irá receber os dados é chamado de componente filho. Uma situação comum para se utilizar props seria quando o usuário não quer repetir todo um código em todas as páginas, então ele irá colocar o código em apenas um componente e caso deseja utilizar aquele código irá chamar esse determinado componente. Um exemplo seria o que foi utilizado na Aplicação Web.

```
1  const PagTitulo = ({ titulo }) => {
2      return (
3          <section className="hero is-small is-primary">
4              <div className="hero-body">
5                  <h1 className="has-text-black is-size-2 is-uppercase
                   has-text-weight-bold">{ titulo }</h1>
6              </div>
7          </section>
8      )
9  }
10
11 export default PagTitulo;
```

Código 4.7: Nesse exemplo ao invés de repetir esse código em todas as páginas foi colocado em apenas um componente chamado PagTitulo que por padrão vai receber uma variável chamada "titulo"com o valor recebido de outro componente.

```
1  function App() {
2    return (
3      <Router>
4        <div className="App">
5         <Navbar />
6         <div className="container is-max-widescreen">
7            <Switch>
8              <Route exact path="/">
9                  <PagTitulo titulo="Página Inicial"/>
```

```
10              </Route>
```

Código 4.8: Agora em outro componente está sendo chamado o componente PagTitulo passando o valor "Página inicial"para o parâmetro titulo no fim essa chamada do componente irá fazer a mesma coisa caso o usuário tivesse digitado aquele código do componente PagTitulo mas dessa forma ele exemplifica além de evitar toda a repetição de um código inteiro

4.7 USEEFFECT HOOK

Assim como os Hooks, o useEffect Hook também foi introduzidos na versão 16.8, sua principal característica é rodar a função a cada vez que os componentes são renderizados.

```
1 useEffect(() => {
2  console.log('use effect');
3  console.log(users);
4 });
```

Código 4.9: Essa função irá rodar toda vez que um componente for renderizado novamente

4.8 USEEFFECT DEPENDÊNCIAS

useEffect dependências permitem ao usuário colocar dependências para quando a função do useEffect irá rodar, essa condição é colocada como um segundo argumento em um colchete. Colchetes vazios significam que a função irá rodar somente na primeira vez que o componente for renderizado. Colchetes com o valor de um State dentro significa que a função irá rodar na primeira vez que o componente for renderizado e quando esse determinado State mudar.

4.9 REACT ROUTE

Route ou Rotas seria a maneira de navegar entre as páginas de um site.

```
1 import { BrowserRouter as Router, Route, Switch } from 'react-router-dom
    '
2 import { Link } from 'react-router-dom';
3
4 function app() {
```

```
5    return (
6      <Router>
7       <div className="App">
8         <Navbar />
9          <Switch>
10           <Route exact path="/">
11               <PagTitulo titulo="Página Inicial"/>
12           </Route>
13           <Route path="/componentes/AdicionarUsuario">
14               <AdicionarUsuario />
15           </Route>
16           <Route path="/componentes/AdicionarImovel">
17               <AdicionarImovel />
18           </Route>
19           <Route path="/componentes/ListaUsuarios">
20               <UsuariosCriados />
21           </Route>
22           <Route path="/componentes/ListaImoveis">
23               <ImoveisCadast />
24           </Route>
25           <Route path="/UserInfo/:id">
26               <DetalhesUsuario />
27           </Route>
28           <Route path="/imovInfo/:id">
29               <DetalhesImoveis />
30           </Route>
31         </Switch>
32      </div
33     <Router>
34 );
35 }
```

Código 4.10: Estrutura de uma Rota.

BrowserRouter: é o escopo onde as rotas serão declaradas, por ser um nome muito extenso é comum ser usado outro nome, como Router.

Route: responsável por renderizar a interface quando o path ser igual ao esperado.

Switch: utilizado quando o usuário adiciona várias rotas, o Switch fará com que apenas a rota específica seja processada, sem o componente Switch todas as rotas serão processadas.

Em Rotas também existem os Routers Links, comumente usados quando o usuário clica em um link dentro de uma página será enviado uma solicitação para o servidor que irá retornar uma página html, com routers links ele irá interceptar essa solicitação e irá carregar a página com o conteúdo específico, deixando a aplicação mais rápida.

Este capítulo apresentou um estudo mais detalhado do framework React, mostrando vários de seus recursos e como funcionam. O próximo capítulo será a apresentação

da aplicação web que foi desenvolvida utilizando o estudo desde capítulo.

5 SISTEMA DE CADASTRO DE IMÓVEIS E USUÁRIOS

5.1 APLICAÇÃO WEB

Após a escolha do framework, foi desenvolvida uma aplicação web para demonstrar os recursos do framework React. A aplicação tem como propósito a criação de usuários e imóveis, além de permitir a visualização e remoção dos imóveis e usuários criados. A aplicação consiste em dez componentes, porém três seriam apenas seções das páginas e não exatamente páginas inteiras como os outros sete componentes. Para a estilização da aplicação foi utilizado o framework Bulma (BULMA, 2022).

5.1.1 Componente Página Inicial

Este componente é responsável por introduzir a aplicação ao usuário, nessa página, conforme é visto na Figura 5.1, o usuário irá decidir o que deseja realizar, se irá cadastrar novos imóveis e/ou usuários ou irá apenas visualizar imóveis e usuários registrados anteriormente.

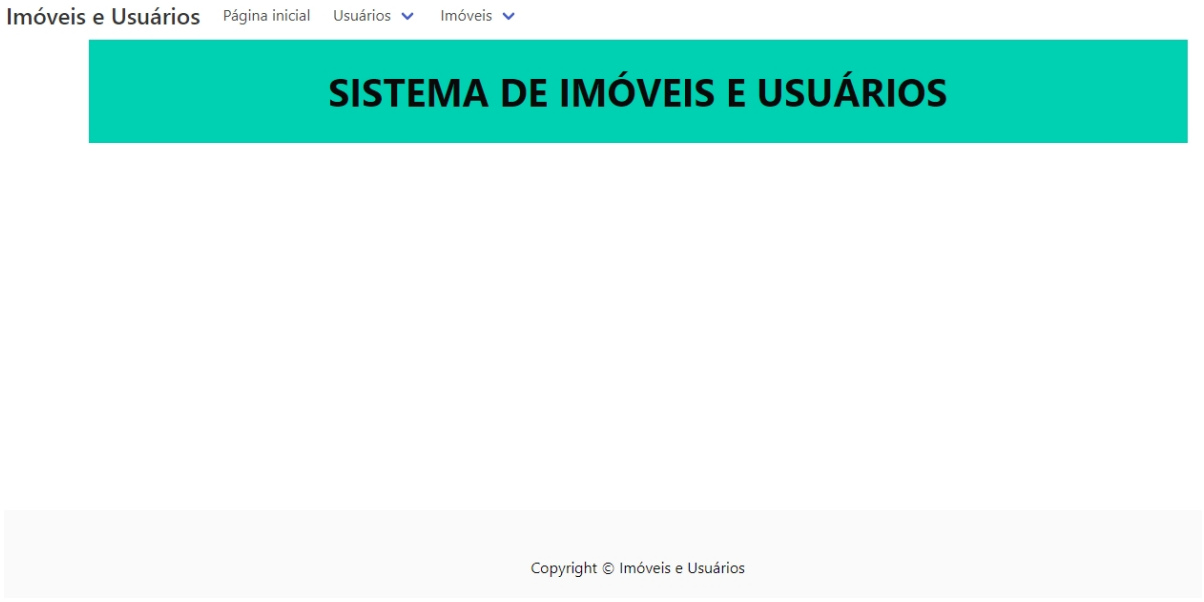

Figura 5.1 – Página Inicial da Aplicação Web.

No desenvolvimento da aplicação de cadastro de imóveis e usuários foi definida uma página inicial, onde foi definido o recurso do framework React chamado *React Route* descrito na Seção 4.9.

A Figura 5.2 apresenta o recurso *React Route* da página inicial da aplicação web desenvolvida para cadastra de imóveis e usuários.

```
15  import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
16
17  function App() {
18    return (
19      <Router>
20        <div className="App">
21          <Navbar />
22          <div className="container is-max-widescreen">
23            <Switch>
24              <Route exact path="/">
25                <PagTitulo titulo="Sistema de imóveis e usuários" />
26                <section className="section is-medium"></section>
27                <section className="section is-small"></section>
28              </Route>
29              <Route path="/componentes/AdicionarUsuario">
30                <section className="section is-small">
31                  <AdicionarUsuario />
32                </section>
33              </Route>
34              <Route path="/componentes/AdicionarImovel">
35                <section className="section is-small">
36                  <AdicionarImovel />
37                </section>
38              </Route>
39              <Route path="/componentes/ListaUsuarios">
40                <section className="section is-small">
41                  <UsuariosCriados />
42                </section>
43              </Route>
44              <Route path="/componentes/ListaImoveis">
45                <section className="section is-small">
46                  <ImoveisCadast />
47                </section>
48              </Route>
```

Figura 5.2 – Código da página inicial da Aplicação Web.

Como pode ser visualizado na Linha 24, o atributo *exact* do *React Route* serve para quando tem várias rotas com urls parecidas, desativando o caminho parcial (*path*) para uma rota e garante que ela só retorne a rota se o caminho for um caminho exato para o URL atual.

Além disso, foram definidas diversas rotas para componentes da aplicação web, conforme descrito na Seção 4.3. Cada rota possui um *path* especificado e uma chamada para o componente, o qual é composto de um conjunto de código que pode ser chamado em diferentes partes da aplicação web.

5.1.2 Componente Cadastro de imóveis

A aplicação de cadastro de imóveis e usuários possibilita um usuário cadastrar novos imóveis, a opção de cadastro pode ser acessada através do menu de navegação da página inicial. Assim que a página é carregada será mostrado um formulário com as informações necessários para um imóvel ser cadastrado. A Figura 5.3 Apresenta a página em si com o formulário que é exibido.

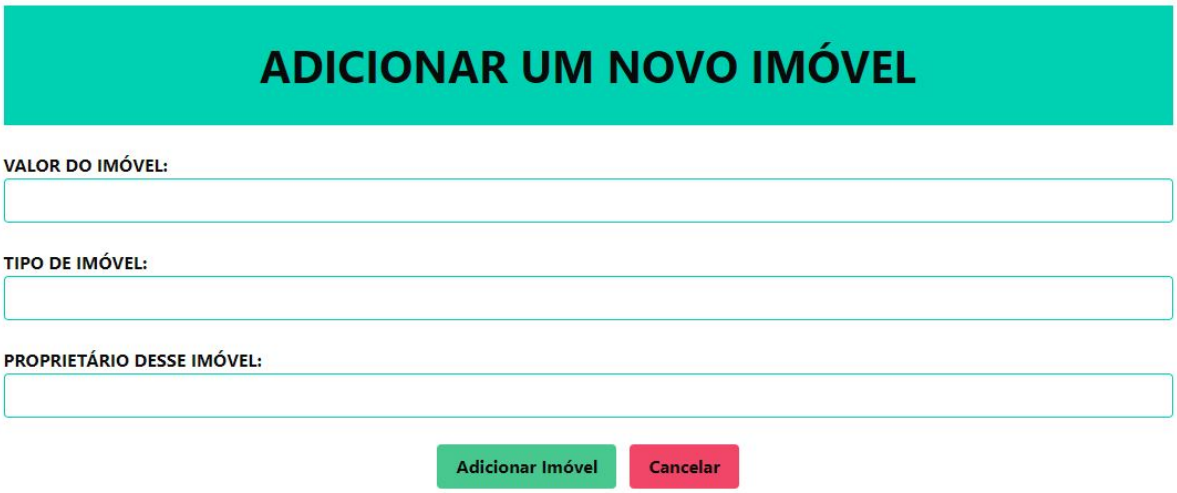

Figura 5.3 – Página de cadastros de imóveis.

Para o desenvolvimento da funcionalidade de cadastro de imóveis foi utilizado o recurso *Hooks* do framework React. Além disso, utilizamos o recurso *preventDefault* para evitar que necessitamos de carregar a página para atualizar as informações adicionadas. Por outro lado, durante o desenvolvimento da aplicação, o formato de texto JSON foi utilizado para hospedar os dados dos imóveis e usuários da aplicação. Por fins de simplicidade, foi utilizado o JSON Server (NPM, 2022) para simular uma API (*Application Programming Interface*) para criar um REST API com recursos de registro e busca de informações em um arquivo de texto em formato JSON.

A Figura 5.4 apresenta o código de cadastro de imóveis.

```
4  const AdicionarImovel = () => {
5    const [valor, setValor] = useState("");
6    const [tipo, setTipo] = useState("");
7    const [proprietario, setProprietario] = useState("");
8    const [isPending, setisPending] = useState(false);
9
10   const handleSubmit = (e) => {
11     e.preventDefault();
12     const imovel = { valor, tipo, proprietario };
13     console.log(imovel);
14
15     setisPending(true);
16
17     fetch("http://localhost:8080/Imoveis", {
18       method: "POST",
19       headers: { "Content-Type": "application/json" },
20       body: JSON.stringify(imovel),
21     }).then(() => {
22       console.log("Um novo imóvel foi cadastrado");
23       setisPending(false);
24     });
25   };
26
27   return (
28     <div className="Adicionar-Imovel">
29       <PagTitulo titulo="Adicionar um novo imóvel" />
30       <form onSubmit={handleSubmit}>
```

Figura 5.4 – Código da página de cadastros de imóveis.

Como pode ser visualizado nas Linhas 5 à 8 está sendo estabelecido e utilizado Hooks, para especificação de valores e variávais para alteração dos atributos de um imóvel. A constante *isPending* foi utilizada apenas para simular o carregamento da página no cadastro. Essa constante pode ser alterada para apresentar para o usuário um sistema de carregamento que simula o processamento de cadastro de um imóvel (Linha 15). Após o encerramento do cadastro de um novo imóvel, a constante é alterada para falso, encerrando o recurso de carregamento da página (Linha 23).

A função handleSubmit será chamada no cadastro de um novo imóvel. Essa função possui código responsável pelo registro de novos imóveis na aplicação. Na Linha 11, o *e.preventDefault();* será usadopara que toda vez que o botão de cadastro seja clicado, ele irá atualizar as informações da página. Na Linha 12, um objeto *imovel* foi criado com os valores pré-definidos pelo recurso *Hooks*.

O cadastro de um novo imóvel é armazenado no arquivo JSON através da chamada do método *fetch*, passando url do arquivo (Linha 17), tipo de requisição (*request*), como é possível visualizar na Linha 18 o *POST*. A seguir, os *headers* são especificados. Na propriedade *body*é necessário transformar para o tipo de formato JSON através da

função *JSON.stringify* passando o objeto imovel. Na Linha 21, após a requisição ter sido concluída, o método *then* irá acionar uma função que retorna uma mensagem de cadastro realizado.

5.1.3 Componente *ListaUsuarios*

Para fins de apresentação de funcionalidades da aplicação desenvolvida neste trabalho, o componente *ListaUsuarios* foi utilizado para apresentar recursos do framework React, o qual é destinado para apresentação de informações sobre os usuários do sistema (Figura 5.5).

USUÁRIOS CRIADOS

ID	Nome	Idade	Endereço	CPF	Opções
2	Pedro	22	Rua exemplo	995.393.332-11	Exibir Informações
3	Marcio	45	Rua teste	854.549.883-99	Exibir Informações
4	Eloisa	28	Rua exemplo	114.358.931-23	Exibir Informações
5	Paulo	34	Rua exemplo	774.573.893-74	Exibir Informações
6	Maria	26	Rua exemplo	867.490.683-57	Exibir Informações
7	Gabriel	28	Rua exemplo	072.759.275-43	Exibir Informações

Figura 5.5 – Página de visualização de usuários disponíveis.

Como pode ser visualizado, uma tabela foi utilizada para apresentação das informações dos usuários do sistema contendo identificador, nome, idade, endereço, CPF e opções para apresentação de mais detalhes.

A Figura 5.6 apresenta o código da página de listagem de usuários da aplicação de cadastro de imóveis e usuários. Alguns recursos do framework React foram utilizados neste componente para apresentação das informações da aplicação, tais como o método *useFetch* para a busca dos dados do arquivo no formato JSON a função *map* que utiliza uma função *Callback* para retornar um novo array de cada usuário do sistema.

```
5   const UsuariosCriados = () => {
6       const { dados: usuario , isPending, erro } = useFetch('http://localhost:8080/Usuarios');
7     return (
8       <div>
9         {erro && <div>{erro}</div>}
10        <PagTitulo titulo="Usuários criados" />
11        {isPending && (
12          <div>
13            <div>Carregando....</div>
14            <progress class="progress is-large is-success" max="100"></progress>
15          </div>
16        )}
17        <div className="UserList">
18          {usuario && (
19            <div>
20              <table className="table is-fullwidth is-hoverable">
21                <thead>
22                  <tr>
23                    <th>ID</th>
24                    <th>Nome</th>
25                    <th>Idade</th>
26                    <th>Endereço</th>
27                    <th>CPF</th>
28                    <th>Opções</th>
29                  </tr>
30                </thead>
31                <tbody>
32                  {usuario.map((UserInfo) => (
33                    <tr>
34                      <th key={UserInfo.id}>{UserInfo.id}</th>
35                      <td key={UserInfo.nome}>{UserInfo.nome}</td>
36                      <td key={UserInfo.idade}>{UserInfo.idade}</td>
37                      <td key={UserInfo.endereco}>{UserInfo.endereco}</td>
38                      <td key={UserInfo.cpf}>{UserInfo.cpf}</td>
39                      <td>
40                        <Link to={`/UserInfo/${UserInfo.id}`}>
41                          Exibir Informações
42                        </Link>
43                      </td>
44                    </tr>
```

Figura 5.6 – Código da página de visualização de usuários.

Como pode ser visualizado na Linha 6, o método *useFetch* é usado passando a URL do arquivo JSON contendo os registros de usuários da aplicação. Como retorno, os dados são atribuídos para a variável *usuario*, a definição de carregamento é inserida na variável *isPending* e a variável *erro* para eventuais mensagens de erro no processo de carregamento das informações.

Caso ocorra algum erro nesse processo, apenas essas informações são apresentadas na página (Linha 9). Por outro lado, enquanto as informações dos usuários estiverem sendo carregadas na variável *dados*, a página apresentará a mensagem “Carregando..."pois a variável *isPending* está definida como verdadeira. Após a conclusão, a variável é alterada para falso e a mensagem de carregamento não será mostrada.

Nesse momento, a variável *usuario* conterá todos os registros de usuários para serem apresentadas na página. Então, é usada a função *map* para *usuario* que utiliza

de uma função *Callback* para retornar um novo array de cada usuário contendo as suas respectivas informações (Linha 32-38).

Por fim, a Linha 40 apresenta o uso do recurso *Link*, apresentado na Seção 4.9, com atributo *to* para permitir que seja possível abrir uma página com maiores informações sobre um determinado usuário.

6 CONCLUSÃO

O desenvolvimento web com a utilização dos frameworks veio para facilitar e ajudar os programadores. Em um mundo com oportunidades no mercado atual aumentando, um framework serve como um melhor amigo para um programador, cada um apresentando suas características, seus pontos positivos e negativos, mas todos entregando uma maneira de ajudar no desenvolvimento web.

Esse trabalho estudou sobre três frameworks, Angular, Vue.js e React. Angular se mostrou um framework complexo, mas muito viável para grandes projetos. O Vue é o caçula entre os três, mas possui um potencial para se tornar mais popular por usuários e grandes empresas. O React possui uma flexibilidade e uma grande procura por empresas, com base nisso esse framework foi escolhido para o desenvolvimento da aplicação web do trabalho.

Nessa aplicação foi usado diversos recursos do React, como JSX para simplificar a sintaxe do código, Hooks para deixá-lo mais dinâmico, Routes para uma navegação simples entre páginas, uso de componentes para organizar a aplicação.

A principal dificuldade encontrada no processo de criação do TCC foi encontrar uma boa fonte de estudos sobre React, pois parecia que em cada site, canal, curso online tinha um jeito diferente de ensinar, mas com ajuda do professor orientador foi possível encontrar um canal no Youtube que explicasse de uma maneira simples e intuitiva.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, V. de. **UM ESTUDO SOBRE PADRÕES E TECNOLOGIAS PARA O DESENVOLVIMENTO WEB–BACK-END**. 2016. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2016.

ANGULAR. **The modern web developer's platform**. 2016. (Acessado em 04 de julho de 2022). Disponível em: <https://angular.io/>.

BANKS, A.; PORCELLO, E. **Learning React: functional web development with React and Redux**. [S.l.]: "O'Reilly Media, Inc.", 2017.

BULMA. **Bulma: the modern CSS framework that just works.** 2022. (Acessado em 24 de abril de 2022). Disponível em: <https://bulma.io/>.

CROCKFORD, D. **JavaScript: The Good Parts: The Good Parts**. [S.l.]: "O'Reilly Media, Inc.", 2008.

DIRIGIDA, F. **Procura por programador cresceu mais de 100% durante a pandemia**. 2020. (Acessado em 25 de abril de 2022). Disponível em: <https://folhadirigida.com.br/empregos/noticias/mercado/vagas-de-programador-cresceram-mais-de-100-durante-a-pandemia>.

ELLIOTT, E. **JavaScript Factory Functions with ES6+**. 2017. (Acessado em 10 de fevereiro de 2022). Disponível em: <https://medium.com/javascript-scene/javascript-factory-functions-with-es6-4d224591a8b1>.

LUONG, Q. Web application development with reactjs framework; case: Web application for an association. 2019.

MOREIRA, T. M. et al. Biblioteca de componentes react inspirados no framework css bulma. Florianópolis, SC, 2020.

NPM. **JSON Server**. 2022. (Acessado em 24 de abril de 2022). Disponível em: <https://www.npmjs.com/package/json-server>.

PAIXãO, J. R. da. **O que é framework? E Por que você precisa de um?** 2017. (Acessado em 10 de fevereiro de 2022). Disponível em: <https://medium.com/desenvolvendo-com-paixao/o-que-\%C3\%A9-framework-e-por-que-voc\%C3\%AA-precisa-de-um-e8b3a47c182f#:~:text=O\%20principal\%20objetivo\%20de\%20um,implementar\%2C\%20dessa\%20maneira\%20o\%20programador>.

PHAN, H. D. React framework: concept and implementation. 2020.

REACT. **Uma biblioteca JavaScript para criar interfaces de usuário**. 2013. (Acessado em 04 de julho de 2022). Disponível em: <https://pt-br.reactjs.org//>.

____. **Introdução aos Hooks**. 2022. (Acessado em 14 de agosto de 2021). Disponível em: <https://pt-br.reactjs.org/docs/hooks-intro.html>.

SAKS, E. Javascript frameworks: Angular vs react vs vue. 2019.

SCHNEIDER, A. H. Desenvolvimento web com client side rendering: combinando single page application e serviços de backend. 2016.

SILVA, M. S. **JavaScript-Guia do Programador: Guia completo das funcionalidades de linguagem JavaScript**. [S.l.]: Novatec Editora, 2010.

STATISTA. **Most used web frameworks among developers worldwide, as of 2021**. 2022. (Acessado em 10 de fevereiro de 2022). Disponível em: <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>.

VUE.JS. **The Progressive JavaScript Framework**. 2014. (Acessado em 04 de julho de 2022). Disponível em: <https://vuejs.org/>.

W3SCHOOLS. **React JSX**. 2022. (Acessado em 22 de junho de 2021). Disponível em: <https://www.w3schools.com/react/react_jsx.asp>.

WILLIAMMIZUTA. **Frameworks e bibliotecas JavaScript: quando e qual usar**. 2017. (Acessado em 25 de abril de 2022). Disponível em: <https://blog.elo7.dev/frameworks-js/>.

WOHLGETHAN, E. **SupportingWeb Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js**. 2018. Tese (Doutorado) — Hochschule für Angewandte Wissenschaften Hamburg, 2018.

ZAKAS, N. C. **Maintainable JavaScript: Writing Readable Code**. [S.l.]: "O'Reilly Media, Inc.", 2012.