BRUNO RIBEIRO MOREIRA

RELATÓRIO DE ESTÁGIO:

DESENVOLVIMENTO FRONT-END NA IOASYS

LAVRAS - MG

2020

BRUNO RIBEIRO MOREIRA

RELATÓRIO DE ESTÁGIO:

DESENVOLVIMENTO FRONT-END NA IOASYS

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Paulo Afonso Parreira Júnior

Orientador

LAVRAS-MG
2020

BRUNO RIBEIRO MOREIRA

RELATÓRIO DE ESTÁGIO:

DESENVOLVIMENTO FRONT-END NA IOASYS

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 02 de setembro de 2020.
Dr. Paulo Afonso Parreira Júnior UFLA
Dr. Rafael Serapilha Durelli UFLA
Nelson Sinis ioasys

Prof. Paulo Afonso Parreira Júnior

Orientador

LAVRAS-MG
2020

AGRADECIMENTOS

À ioasys, pela concessão do estágio.

À Universidade Federal de Lavras, especialmente ao Departamento de Ciência da Computação, pela oportunidade.

Ao professor Dr. Paulo Afonso Parreira Júnior, pela orientação, paciência e disposição para ajudar.

[...] A todos funcionários do DCC/UFLA.

Aos meus pais, em especial minha mãe Roberta Pace Ribeiro pelo amor e apoio incondicional, em todas as minhas decisões nas diferentes etapas da minha vida e aos meus amigos e colegas.

À Raíssa Zanetti da Silva, pelo companheirismo e apoio em todos os momentos e singular torcida.

MUITO OBRIGADO!

RESUMO

O presente trabalho tem como objetivo descrever as atividades realizadas durante o período de estágio na empresa Innovation Oasys – ioasys. Atualmente com sede em Belo Horizonte – MG e operando em diversas regiões do Brasil, a ioasys tem como principal atividade a solução tecnológica de ponta a ponta, ou seja, desenvolvimento de aplicativos com estudo de mercado e lançamento. Durante o período de estágio, foi desenvolvido diversos projetos para clientes da ioasys, utilizando tecnologias JavaScript, HTML e CSS com ReactJS para o front end. As atividades foram desenvolvidas utilizando a metodologia ágil Scrum.

Palavras-chave: ReactJS, JavaScript, HTML, CSS, Scrum

ABSTRACT

This work aims to describe the activities carried out during the internship period at the company Innovation Oasys - ioasys. Currently based in Belo Horizonte – MG, operating in several regions of Brazil, the main activity of ioasys is the end-to-end technological solution, that is, application development with market research and launch application. During the internship period, several projects were developed for ioasys clients, using JavaScript, HTML and CSS technologies with ReactJS for the front end. The activities were developed using the agile Scrum methodology.

Keywords: ReactJS, JavaScript, HTML, CSS, Scrum

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A *innovation oasys* (ioasys) é uma empresa fundada em 2012, que possui sua sede em Belo Horizonte e outras unidades nos municípios de Lavras (MG) e São Paulo (SP). A empresa atua com soluções tecnológicas de ponta a ponta, desde a produção do software, até a publicação do mesmo nas lojas de aplicativos ou na web. Atualmente, a empresa conta com um time de, aproximadamente, 170 colaboradores.

Durante a realização do estágio na empresa, foram desenvolvidos projetos com a tecnologia *ReactJS*[1], uma biblioteca mantida pelo Facebook para soluções de aplicações *front-end*[2]. Anteriormente ao desenvolvimento dos projetos, foi oferecido aos estagiários um treinamento, com duração de um mês, para aprendizado e familiarização com a tecnologia e dos padrões de codificação utilizados na empresa. Além do *ReactJS*, também foram utilizadas as linguagens HTML *(Hypertext Markup Language*), CSS (*Cascading Style Sheets*) e *JavaScript*. Esses conceitos são mais bem detalhados no referencial teórico deste trabalho.

Dentre os projetos desenvolvidos pelo estagiário, e que são apresentados neste trabalho, o pioneiro foi para o cliente ABC[3], que é uma empresa de consórcios. Outro projeto desenvolvido foi para o cliente XYZ, no qual houve a adição de *features* e manutenção da página administrativa do app do cliente. Além desses projetos, foram aplicados outros desafios no período de treinamento, como por exemplo, a criação de uma página para um sistema da empresa e, também, o desenvolvimento de um quadro de *commits* do *GitHub*.

No Capítulo 2 deste trabalho, é apresentada a tecnologia *ReactJS*, com as principais bibliotecas utilizadas nos projetos desenvolvidos durante o estágio. Além disso, também será discutido sobre as principais linguagens de programação envolvidas no *ReactJS* e consequentemente nos projetos desenvolvidos. Em sequência, no Capítulo 3, são discutido os projetos e atividades desenvolvidas pelo autor deste relatório de estágio. Ao final desse relatório de estágio (Capítulo 4), tem-se as considerações finais, apontando as experiências adquiridas, as dificuldades

[1] Encontrada em https://pt-br.reactjs.org
[2] Responsável pela parte da interface gráfica do software, que interage diretamente com o usuário.
[3] Por questões de sigilo, o nome real dos clientes foi substituído por um nome fictício.

encontradas, ressaltando os pontos negativos e positivos dessa vivência, comparando o trabalho no âmbito laboral e no âmbito domiciliar, devido a pandemia do Covid-19.

2. REFERENCIAL TEÓRICO

Neste capítulo, são abordadas as principais tecnologias e os conceito utilizados nos projetos desenvolvidos ao longo do estágio.

2.1 Framework

A definição do que é um framework está presente em diversos lugares e de várias formas. Entretanto, essas definições convergem para um ponto principal, que é o de reuso do *design* ou do código de um sistema de software. Gerdessen (2007) afirma que *"um framework é um esqueleto para aplicações, visando um domínio específico, abrangendo certas áreas-chave e a interação entre elas dentro desse domínio específico".*

Na literatura, o significado de framework *JavaScript* pode variar, dependendo da circunstância. De modo geral, o termo framework é utilizado quando o controle do fluxo de execução da aplicação passa e ser de responsabilidade do framework, e não do desenvolvedor. Assim, separam-se os frameworks das bibliotecas *JavaScript*, pois uma biblioteca oferece uma coleção de funções que assistem à execução da aplicação, enquanto que um framework gerencia todo o seu fluxo de dados e de execução da mesma (VOUTILAINEN, 2014). Um exemplo de biblioteca *JavaScript* é o *ReactJS,* que será tratado posteriormente neste documento. Já um exemplo de framework é o Node.js, que é um ambiente de execução *JavaScript* do lado do servidor. Isso significa que, com o *Node.js*, é possível criar aplicações *JavaScript* para rodar como uma aplicação *standalone*, ou seja, sem a necessidade de um browser para sua execução.

2.2 Hypertext Markup Language (HTML)

Dentre as tecnologias utilizadas para desenvolvimento de páginas *web*, tem-se a linguagem HTML *(Hypertext Markup Language*). HTML é a linguagem que define os elementos de uma página *web* por meio de *tags* e atributos. Robbins (2012) afirma que HTML não é uma linguagem de programação, mas sim uma linguagem de

marcação, ou seja, ela especifica uma forma de descrever vários tipos de componentes de um documento, como títulos, parágrafos e listas.

Pode-se observar, na Figura 1, um exemplo básico de uso da linguagem HTML, no qual tem-se a *tag* de abertura <html> e fechamento </html>. Logo, tudo inserido entre essas *tags* integra a página HTML. Existem subdivisões entre o início e o final da página HTML, como por exemplo o *header* e o *body*. O *header* é responsável pelo cabeçalho da página, enquanto o *body* compõe o corpo, no qual fica todo o restante do conteúdo com as demais *tags* HTML.

Figura 1 - Exemplo de "Olá Mundo!" em HTML

Fonte: Autor

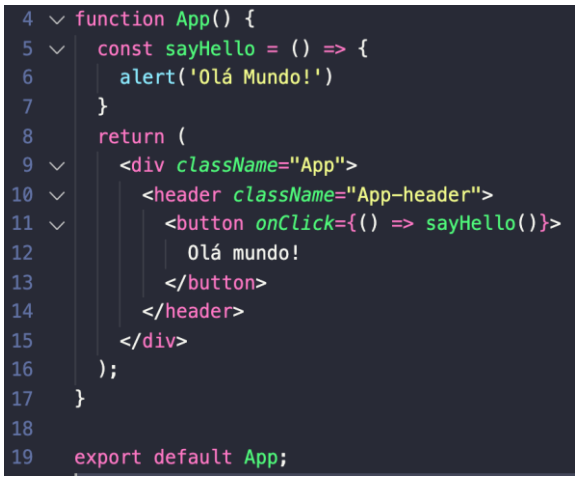
2.3 JavaScript

O americano Brendan Eich foi o programador pioneiro na linguagem *JavaScript*, em 1995, criando-a em apenas dez dias. Ela foi lançada, inicialmente, com o nome Mocha, em homenagem ao fundador da Netscape. No mesmo ano, o nome foi alterado para *JavaScript*, tendo em vista que a Netscape queria anunciar a parceria feita com a Sun Microsystems, empresa criadora da linguagem de programação Java. O objetivo principal era gozar de toda publicidade e notícias na imprensa. O novo nome levantou muitas especulações, já que cria a ilusão de que essa nova linguagem seria baseada em Java, o que não é verdade (SEVERANCE, 2012).

JavaScript (JS) é muito utilizada no âmbito da programação web, sendo suportada por praticamente todos os navegadores. O objetivo da linguagem é permitir fazer verificações ou mudar dinamicamente a apresentação visual das páginas, conforme as ações executadas pelo usuário (DORNELLES, 2016). A Figura 2 exemplifica o uso da linguagem JS. Nesse código, ao clicar sobre o botão, o usuário

irá disparar a execução de uma função JS que irá alertar o usuário com a frase “Olá Mundo!”.

Figura 2 - Exemplo do uso da linguagem JavaScript



Fonte: Autor

2.4 Cascading Style Sheets (CSS)

A linguagem CSS (*Cascading Style Sheets*) utiliza uma marcação existente (por exemplo, um código HTML) para personalizar a apresentação de seus elementos. Em outras palavras, CSS é responsável pela aparência dos componentes de uma página, adicionando cores, espaçamento, entre outros (LEWIS; MOSCOVITZ 2010).

A estilização de um botão com código CSS pode ser vista na Figura 3, na qual é definida a largura, altura, cor de fundo e borda de um botão. No código à direita da Figura 3, é possível observar que o botão possui uma propriedade chamada *className*, cuja função é vincular o código CSS ao botão em HTML.

Figura 3 - Botão estilizado em CSS

Fonte: Autor

Já na Figura 4, é possível observar o resultado do código, isto é, o botão com tamanho definido, a cor de fundo definida e bordas arredondadas.

Figura 4 – Exemplo de botão

Fonte: Autor

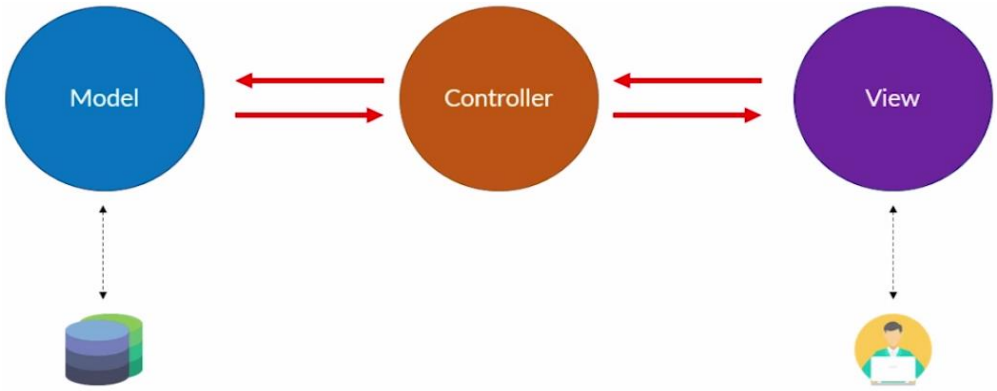
Em resumo, HTML serve para estruturar a informação, CSS serve para estilizar esse conteúdo e *JavaScript* serve para definir o comportamento dos componentes.

2.5 ReactJS

O ReactJS é uma biblioteca *front-end* baseada na linguagem *JavaScript*, lançada em 2013 e mantida pelo Facebook. Ela permite o desenvolvimento de interfaces baseadas em componentes para aplicações *web*. O Facebook adota o ReactJS vigorosamente em seus sites (Facebook, Instagram e WhatsApp), mas empresas como Yahoo, Airbnb, Sony também a utilizam (SCOTTI, 2019).

ReactJS é uma biblioteca *JavaScript* que trabalha com o padrão arquitetural *Model-View-Controller* (MVC). O modelo MVC consiste na separação da aplicação em três camadas: o *Model*, a *View* e o *Controller*. A primeira camada é onde fica implementada e encapsulada toda estrutura lógica da aplicação. A segunda é responsável por disponibilizar as interfaces de interação com o usuário, requerendo dados do *Model*, quando necessário. Já a terceira desempenha a comunicação entre a *View* e o *Model* (SCOTTI, 2019).

Figura 5 - Modelo MVC



Fonte: https://www.devmedia.com.br/guia/asp-net-mvc/38190. Acessado em Junho de 2020.

2.5.1 ReactDOM, Styled Components e React Router

Ao desenvolver uma aplicação com *ReactJS*, não é apenas necessário usar a biblioteca em si; é preciso usar também o ReactDOM. Segundo Cunha (2019), ReactDOM é uma biblioteca que *"deve ser usada para renderizar a interface de usuário no navegador"*. Além disso, de acordo com Banks e Porcello (2017), ela fornece "*métodos específicos do DOM, que podem ser usados no nível superior para sair do modelo do React quando necessário"*.

O DOM (*Document Object Model*) é uma representação lógica de qualquer página web. Ou seja, o DOM é uma estrutura em forma de árvore que contém todos os elementos e as propriedades de um site como nós. O DOM fornece uma interface que permite acessar e atualizar o conteúdo de qualquer elemento de uma página. O ReactDOM fornece os métodos específicos do DOM para serem usados no nível superior da aplicação, permitindo assim um gerenciamento mais eficiente dos elementos DOM da página *web*. Um de seus métodos é o *render,* que pode ser visualizado na linha 7 da Figura 6; esse método é responsável por renderizar o componente passado por parâmetro na página *web*, neste caso, o componente *App*.

Figura 6 - Exemplo de uso do ReactDOM

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  import * as serviceWorker from './serviceWorker';
6
7  ReactDOM.render(
8    <React.StrictMode>
9      <App />
10   </React.StrictMode>,
11   document.getElementById('root')
12 );
```

Fonte: Autor

Outro conceito importante, quando se trabalha com *ReactJS*, é o de *Styled-components*. Segundo Aho (2020), *Styled-components* são uma biblioteca de componentes do *ReactJS*, a qual permite escrever CSS diretamente no arquivo *JavaScript*, por meio do JSX (extensão de sintaxe para *JavaScript*).

Um exemplo da utilização de *Styled-components* é a representação em código visto nas Figuras 7 e 8, sendo que na Figura 7 tem-se o arquivo que define todas as propriedades de estilo, as quais são armazenadas em um componente chamado *Container*. Já na Figura 8, especificamente na linha 4, destaca-se a importação do componente *Container* com suas estilizações. A utilização desse componente se inicia na linha 8 pela abertura da *tag* <*Container*> e sua finalização pela *tag* </*Container*>. Logo, tudo que está entre essas *tags*, terá os estilos aplicados.

ReactJS é uma das bibliotecas *JavaScript* mais utilizadas para desenvolvimento de *Single Page Applications (*SPA). SPA é uma aplicação *web* que roda em uma única página, se assemelhando a um aplicativo *desktop* ou um *mobile*. Ao invés de recarregar toda a página ou redirecionar o usuário para uma página nova, apenas o conteúdo principal é atualizado de forma assíncrona, mantendo toda a estrutura da página estática. Sendo assim, em uma SPA tudo acontece na mesma página e, portanto, é necessário estabelecer um processo de roteamento que define os *endpoints* para as requisições dos seus clientes. Para isso, foi utilizado o *React Router.* Para Souto (2018), *React Router* é um conjunto de componentes e/ou elementos de navegação que integram uma aplicação, facilitando a navegação por links, sem exigir o recarregamento da página.

Figura 7 - Estilização de componente pelo *Styled-components*

```
import styled from 'styled-components';

export const Container = styled.div`
        width: 100%;
        max-width: 1168px;
        display: flex;
        justify-content: space-between;
        align-items: center;
        z-index: 10;
        padding: 0 24px;
        button {
          text-transform: uppercase;
          font-weight: bold;
          width: 240px;
          background: #73678a;
          border-width: initial;
          border-style: none;
          border-radius: 5px;
          cursor: pointer;
          height: 54px;
          transition: background 0.2s;
          font-size: 16px;
          color: white;
          &:hover {
            background: #795a8b;
          }
        }
      `;
```

Fonte: Autor

Figura 8 - Componente estilizado com *Styled-components*

```
import React from 'react';
import logo from '../../assets/images/logo.png';

import { Container } from './Style'

const Header = () => {
    return (
        <Container>
        <img src={logo} alt="Logo" />
        <button type="button"> Acessar </button>
        </Container>
    )
}

export default Header;
```

Fonte: Autor

Na Figura 9 é possível ver a estruturação dos componentes para roteamento no *React Router*. *BrowserRouter* é um componente responsável por informar para a aplicação que a partir dele é chamado o roteamento de componentes. O *Switch* é um componente que recebe vários outros componentes do tipo *Route.* Cada *Route* é uma rota, para a qual deve-se informar o caminho da URL, atributo *path*, e o componente que será renderizado, atributo *component.*

Figura 9 - Roteamento de páginas com React Router Dom

```
import { BrowserRouter, Switch, Route } from 'react-router-dom'

ReactDOM.render(
  <BrowserRouter>
    <Switch>
      <Route path="/" exact={true} component={App} />
      <Route path="/sobre" component={Sobre} />
    </Switch>
  </ BrowserRouter>
  , document.getElementById('root'));
registerServiceWorker();
```

Fonte: Autor

2.5.2 FLUX e React Redux

Em sistemas de grande escala, o fluxo de dados bidirecional pode se tornar muito trabalhoso e de difícil manutenção, pelo fato de diversas *views* comunicarem com diversas *models*. Dessa forma o Facebook criou o Flux, um novo padrão de arquitetura de software unidirecional.

Com a constante mudança de dados em uma aplicação o padrão Flux torna-se importante por pregar a ideia de *single source of truth,* ou seja, única fonte de verdade. Isso significa que existe apenas um lugar que representa o estado do aplicativo e a interface do usuário a utiliza como parâmetro.

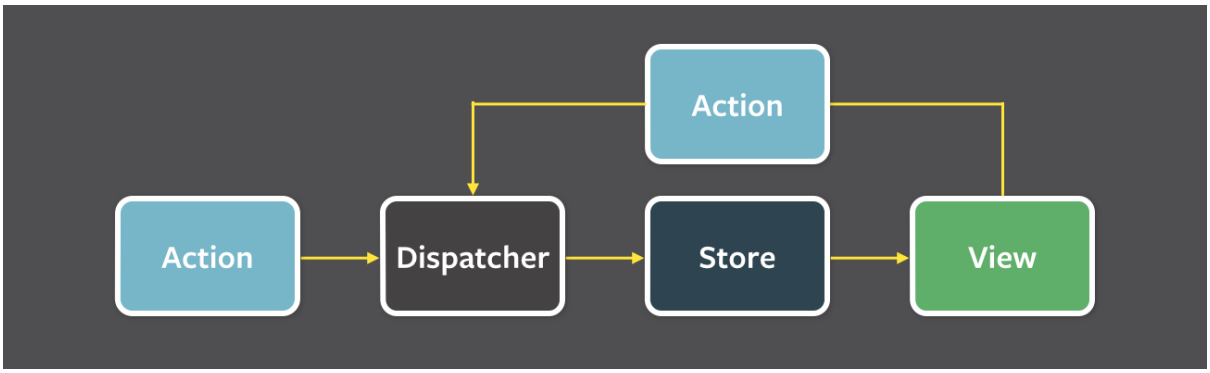
A interface do usuário muda apenas se os dados neste local forem alterados. Dados estes que ficam armazenados na *Store,* sendo que a *Store* é um objeto *JavaScript,* a qual possui todos os estados dos componentes da aplicação.

Já as *Actions* são responsáveis por requisitar algo para a *Store* e são sempre funções puras, ou seja, enviam somente os dados sem funções com efeitos colaterais. Assim, enviam um estado da aplicação para a *Store*, através do *Dispatcher*. O *Dispatche*r recebe a *Action* e a encaminha para a *Store* requisitada, sendo que, em aplicações de larga escala, é comum ter a *Store* dividida em vários *Reducers*. O *Reducer* que faz parte da Store, recebe o estado atual, mais, uma Action. Em seguida, retorna um novo estado.

Como o próprio nome sugere, o Flux permite um fluxo de dados em um único sentido para a comunicação das páginas para gerar respostas ao cliente. Ele complementa os componentes de visualização do *ReactJS* utilizando um fluxo de dados unidirecional, ou seja, evita que várias *views* comuniquem com várias *models*. É mais um padrão do que um framework formal. (FACEBOOK-INC, 2014)

O padrão de arquitetura é formado por uma série de interações entre componentes, interações estas que permitem a comunicação e o fluxo de dados. É formado por uma *Action,* chamada pela *View*, que dispara uma ação levando uma informação através do *Dispatcher* para a Store, o *Dispatcher* solicita a alteração na *Store,* a qual resulta na mudança da *View*, visto abaixo na figura 10 (MARTTILA, 2016).

Figura 10 - Modelo Flux



Fonte: https://facebook.github.io/flux/docs/in-depth-overview/ Acessado em junho de 2020.

O *Flux* pode ser exemplificado pelas figuras 11 e 12, sendo que, na figura 11, é disparado um *Dispatcher,* o qual é responsável por trazer os dados necessários para apresentação na View de comunidades do *Backend*, seguindo a estrutura do *Flux*. Já na Figura 12, na linha 54 e 55 um *Dispatcher* dispara uma *Action* levando a informação que começou a tentativa de comunicação com o *Backend*. Após ser executada a função de comunicação com a API, é disparado mais uma *Action* na linha 62 que, dessa vez, é encarregada por encaminhar os dados que foram recebidos pelo *backend* para a *Store*, assim a *View* requisita a *Store* os dados para poder disponibiliza-lo na página web.

Figura 11 - Chamada de um Dispatcher

```
useEffect(() => {
  if (filterParams !== '') {
    dispatch(filterCommunity(filterParams, pageNow, search));
  } else {
    dispatch(fetchCommunity(search, pageNow));
  }
}, [search, pageNow, dispatch]);
```

Fonte: Autor

Figura 12 - Exemplo de uso do Flux

```
53  export const fetchCommunity = (title, page) => dispatch => {
54    dispatch(communityAction.changeStatusCommunity(false));
55    dispatch(communityAction.defaultStateCommunity());
56    CommunityApi.index(title, page).then(response => {
57      if (response instanceof ErrorMessage) {
58        return dispatch(
59          communityAction.message('Ocorreu um erro, procure o administrador do sistema!')
60        );
61      }
62      dispatch(communityAction.receiveCommunityList(response, false));
63      return null;
64    });
65  };
```

Fonte: Autor

"O padrão de arquitetura é semelhante ao MVC onde o Dispatcher realiza o papel do *Controller*, e o *Store* realiza o papel do *Model*. Porém o Flux dita como são realizadas as interações entre os pacotes, além de possuir componentes independentes e separados". (MARTTILA, 2016)

O *Redux* também compõem-se como uma das bibliotecas disponíveis para o uso com *ReactJS* com uma *Application Programming Interface* (API) simples, API é conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por outros aplicativos. Segundo Fonseca; Balbino (2019), o *Redux* atua com o conceito de programação funcional, criando um estado geral para os componentes do *ReactJS* consumirem essas informações. Fonseca; Balbino trazem ainda que o *Redux* pode ser utilizado junto com *plugins* (como o *redux-thunk*), a fim de auxiliar o gerenciamento de formulários, ações múltiplas, *promises* e ações assíncronas.

O *Redux* é responsavel por gerenciar o estado do componente e disponibilizar para a aplicação. Dessa forma, as *Actions* são responsáveis por chamar os *reducers*. Os *reducers*, por sua vez, gerenciam as informações do estado e comunicam como reagir à determinada *Action*. Já a *Store* armazena todas as informações do estado. (LINS, 2019)

Para Cunha (2019) o *Redux* é considerado eficaz, já que conecta os componentes *ReactJS* com o Servidor, levando a informação para que o servidor armazene no banco de dados, ou requisitando dados.

Outra função do *Redux*, citada por Ribeiro (2019), é a capacidade de *“centralização do state e a lógica da aplicação”*. Proporcionando armazenamento de todas variáveis (podendo ser um campo de texto, botão ou outro componente) e disponibilizando a opção refazer/desfazer, respectivamente.

2.6 Metodologia *Scrum*

O *Scrum* é uma das metodologias ágeis de desenvolvimento de software por etapas, a qual prevê o foco na equipe desenvolvedora, nas entregas que agreguem valor ao cliente e nas respostas a mudanças. No Scrum, os projetos são divididos em ciclos de aproximadamente 10 dias úteis, denominados *Sprints*. A ideia por detrás disso é agilizar e melhorar o fluxo de trabalho e desenvolvimento de uma equipe. As principais atividades do *Scrum* são: (i) *sprint planning,* a qual divide as tarefas de desenvolvimento e pontua, com base em um sistema de classificação, cada tarefa, de acordo com sua complexidade; (ii) *daily scrum* é uma reunião diária com a equipe de desenvolvimento, cujo objetivo é melhorar a comunicação entre toda a equipe, por meio do relato das atividades desenvolvidas desde a última reunião e dos possíveis impedimentos encontrados pela equipe; e (iii) *sprint retrospective*, que trata-se de uma reunião na data final do *sprint* para analisar as últimas atividades desenvolvidas e estudar possibilidades de melhoria para o fluxo de trabalho.

Scrum define os seguintes papéis: *Scrum Master*, *Product Owner* e *Scrum Team*.

> “O *Scrum Master* trabalha para que o processo *Scrum* aconteça e para que não existam impedimentos para os membros da equipe realizarem seu trabalho. Remover os obstáculos apontados na reunião de *Scrum* diária é seu dever, de modo que os desenvolvedores se concentrem apenas nas questões técnicas. Esses obstáculos são colocados em uma lista chamada de *Backlog* de Impedimentos, que fica à vista de todos.” (CARVALHO; MELLO, 2012)

Para Carvalho; Mello (2012), o Dono do Produto (*Product Owner*) é um membro da equipe que representa o cliente (interno ou externo). O *Product Owner* define o grau de importância e prioridade dos requisitos, para isto, é importante que

este membro domine as regras de negócio do cliente. O *Scrum Team* é composto pela equipe desenvolvedora do projeto, responsável por realizar as tarefas definidas na *Sprint Planning*, com total autonomia de execução, desde que corresponda às diretrizes estabelecidas no projeto. Além disso, o *Scrum Team* ainda se responsabiliza pela apresentação do que foi desenvolvido para o *Product Owner*, ou até mesmo diretamente para o cliente.

3. Atividades desenvolvidas

Neste capítulo, são relatadas as atividades desenvolvidas ao longo do estágio. Alguns desses projetos estão disponíveis no GitHub[4], outros são privados. Na empresa ioasys, a revisão de código é realizada por meio de *Pull Requests (PR)*. O *Pull Request* é uma funcionalidade implementada em plataformas de versionamento de código, como o GitHub, na qual um desenvolvedor pode gerar uma notificação que sinaliza a conclusão do desenvolvimento de uma *feature*. Dessa forma, no PR ficam evidentes as alterações feitas no código, que podem ser comparadas com a versão anterior do mesmo. Para o PR ser aprovado, é necessário que seja revisado por outros 2 desenvolvedores da mesma tecnologia. Essa prática ajuda a manter a qualidade do código, como também favorece o aprendizado de novas formas de se resolver determinados problemas de programação.

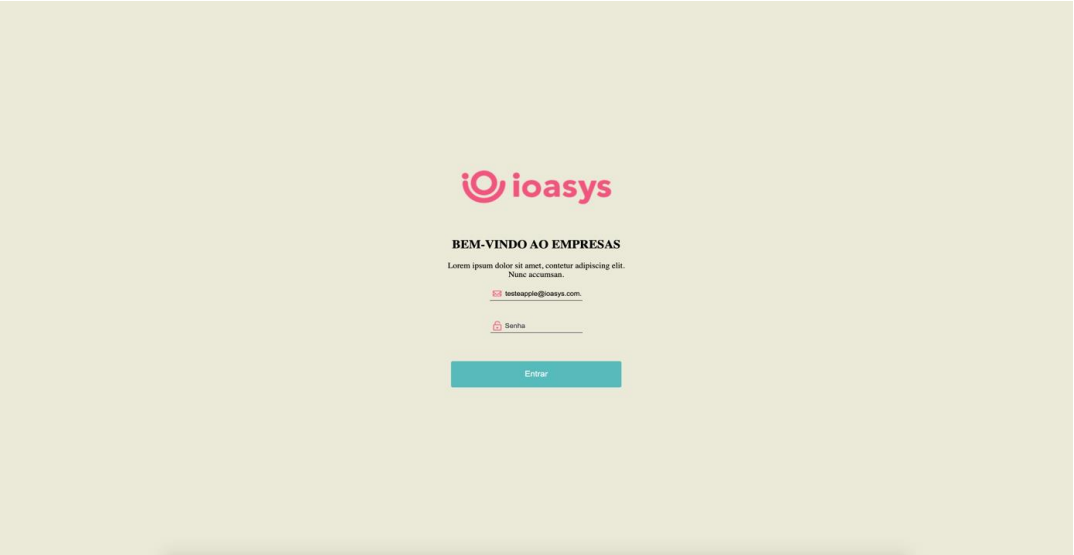
3.1 App Empresas

Como boas-vindas à empresa, o primeiro projeto a ser executado é o *App Empresas*, construindo em *ReactJS*, utilizando diversas bibliotecas e técnicas que são usadas em todos os projetos da empresa. O projeto também serve para se adequar aos padrões de código da empresa, bem como ao cotidiano de revisão de códigos, que são práticas fundamentais no dia a dia do desenvolvedor.

O maior desafio, sem dúvidas, foi a familiarização com a estrutura do projeto e com a biblioteca *ReactJS*. Apesar das dificuldades, foi possível aprender as funcionalidades básicas, como desenvolver uma tela de login com autenticação (Figura 13), realizar buscas por meio de uma API (*Application Programming Interface*), implementada no *Backend*, e realizar a navegação entre telas em numa aplicação *ReactJS*.

[4] Github com os projetos executados está disponível em https://github.com/brunim1101

Figura 13 – Tela de Login - App Empresas



Fonte: Autor

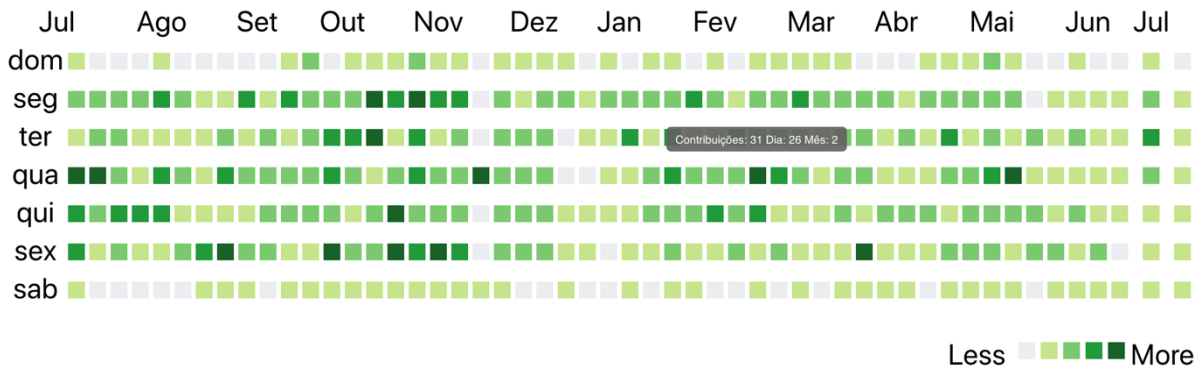
3.2 Calendário de *Commits*

Após a conclusão do *App Empresas*, o mentor responsável pelo estagiário apresentou um novo desafio: elaborar um calendário de *commits* do *GitHub*. Esse calendário é utilizado pelo próprio *GitHub* e disponibilizado na página de perfil de um usuário no *GitHub*, sua principal função é informar a atividade de determinado usuário com projetos mantidos na plataforma.

Um *commit* é uma contribuição de código a um projeto. A proposta do calendário é mostrar, em categorias, o número de *commits* de um determinado projeto no último ano. Além disso, esse calendário deve possuir um esquema de cores, de forma que, quanto mais *commits* em um dia, mais escura é a sua cor, o que pode ser melhor visualizado na Figura 14 – esse tipo de calendário é conhecido como calendário térmico e é adotado pelo *GitHub* na página de perfil de um usuário. Uma das funcionalidades do calendário térmico é que, ao passar o mouse sobre os quadrantes coloridos, mais informações são disponibilizadas sobre o dia, como o número exato de *commits*.

O principal objetivo desse projeto foi aprimorar o conhecimento sobre *ReactJS* de forma a criar um componente que pudesse ser usado em outros projetos. Esse desafio foi mais complexo do que o *App Empresas*, pois envolvia manipulação de datas e requisições de dados a uma API externa, entre outras coisas.

Figura 14 - Calendário de *Commits do projeto ReactJS*, *mantido pelo Facebook no GitHub*



Fonte: Autor

Mais uma vez, a tecnologia utilizada neste desafio foi o *ReactJS*. O desenvolvimento deste desafio foi desempenhado apenas pelo próprio autor deste relatório. A maior vantagem desse projeto foi o aprendizado da manipulação de datas e trabalhar com a API externa do *GitHub*.

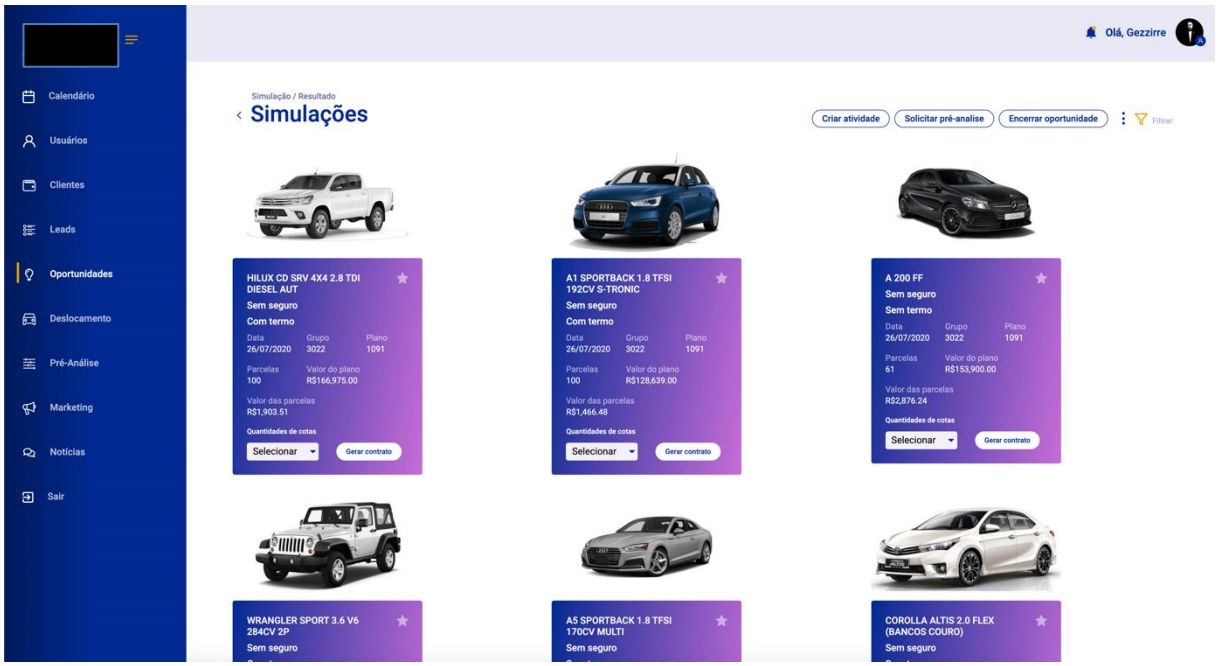
3.3 Cliente ABC

O cliente ABC é uma empresa que possui vários setores de atividades, entre eles, o de consórcio. A ioasys desenvolveu um site e aplicativo Android para ser usado como ferramenta de trabalho por seus consultores. Neste projeto, foram implementadas diversas *features*, como um sistema de buscas e simulação personalizadas de cartas de crédito seguindo vários critérios, conforme pode ser visto na simulação da Figura 15.

Foi desenvolvida, também, uma tela de gerenciamento de calendário com marcações de eventos, por meio do qual era possível marcar visitas para clientes e uma tela para gerenciamento de consórcios, criados pelo simulador de consórcio. Um dos grandes empecilhos desse projeto foi em relação ao próprio cliente, pois o mesmo, após solicitar determinada funcionalidade, solicitava, em seguida, uma segunda função, a qual entrava em conflito com as regras de negócio da primeira. Esse fato gerou grande retrabalho de todo projeto e mobilização de toda a equipe para ajuste de regras de negócio.

Com esse projeto, foi possível ter um aprendizado profundo a respeito de componentes reutilizáveis no *ReactJS*, como por exemplo, um componente de modal que era utilizado em todas as partes do site, alterando somente o conteúdo presente e a funcionalidade do botão.

Figura 15 - Simulação de consórcios



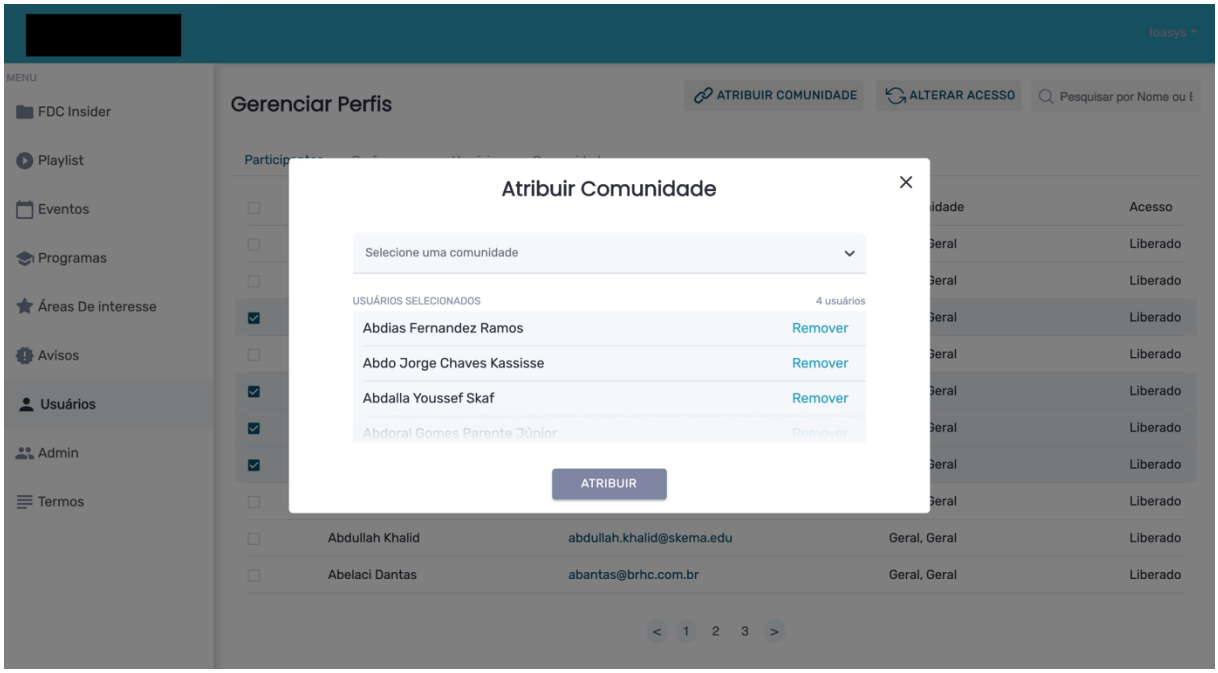
Fonte: Autor

3.4 Cliente XYZ

O cliente XYZ é uma grande escola de negócios (*business*) da América Latina e esse projeto era composto por um aplicativo para Android e iOS e um site para o administrador desse aplicativo mobile. No site, o administrador pode controlar todo o conteúdo do aplicativo para celular, como por exemplo, fazer publicações de conteúdos que podem ter diversos tipos de arquivos, por exemplo, vídeo, áudio, foto ou links, aliado ao texto do próprio conteúdo. Foi desenvolvida, também, uma *feature* que permitia ao administrador pré-visualizar como o conteúdo seria exibido nos aplicativos dos usuários finais. O aplicativo mobile é direcionado para os clientes da XYZ; trata-se de uma espécie de rede social para compartilhamento de conteúdo personalizado de negócios.

O maior desafio desse projeto foi a criação de funções no *JavaScript* para manipulação e controle de dados, tendo em vista que, para diversos fluxos, era necessário transmitir vários dados e os manipular para poder fazer alterações em diversos usuários de uma vez. Na Figura 16 é possível ver um exemplo de fluxo que ocorre na aplicação, no qual, após selecionar usuários em uma tabela de usuários, é possível atribuir comunidades a eles.

Figura 16 - Fluxo seleção de usuários



Fonte: Autor

4. Considerações Finais

O valor de um estágio está muito além do financeiro. É uma experiência enriquecedora em diversos sentidos, principalmente, para o crescimento na carreira e amadurecimento das convicções profissionais. Além de agregar valor aos currículos dos recém-formados, o estágio abre novas portas e aprimora as habilidades iniciadas e desenvolvidas na faculdade, como habilidades técnicas e interpessoais. Ele serve ainda para demonstrar aos potenciais empregadores o grau de comprometimento desses futuros profissionais.

Ao final do estágio, conclui-se que a utilização das boas práticas adquiridas durante o período de graduação é uma forma de reduzir o retrabalho, pois as más práticas aumentam de forma considerável os custos nas etapas de manutenção e evolução do sistema. Muitas disciplinas foram importantes para que o desenvolvimento das atividades do estágio fossem realizadas de maneira eficiente e eficaz. Dentre elas, estão as disciplinas de programação, como Algoritmos e Estruturas de Dados e Programação Orientada a Objetos. Tais disciplinas permitem ao aluno desenvolver o raciocínio lógico, para que o mesmo possa, então, desenvolver suas habilidades na linguagem mais apropriada para a resolução do problema.

Outras disciplinas importantes de se mencionar são Engenharia de Software e Processos de Software, que fornecem um embasamento teórico adequado sobre o desenvolvimento de sistemas orientado pelas melhores práticas dessas disciplinas. A maioria dos sistemas computacionais exige um meio de armazenamento dos dados. Assim sendo, as disciplinas Banco de Dados 1 e 2 tiveram grande importância para concretização das atividades do estágio. Em geral, as bases de dados dos sistemas reais possuem grandes volumes de dados e, muitas vezes, consultas complexas devem ser realizadas nestas bases. Assim, a falta dos conhecimentos adquiridos em tais disciplinas poderiam impactar negativamente na conclusão desse tipo de atividade.

A realização do estágio proporcionou ao estagiário conhecimento sobre o mercado de trabalho, por exemplo sobre as técnicas e ritos que pertencem ao cotidiano de um desenvolvedor no mercado de trabalho. Devido à pandemia gerada pelo Covid-19, foi possível viver dois momentos no mercado de trabalho, o trabalho presencial e o trabalho remoto. Do trabalho presencial, é possível destacar positivamente o contato direto nas reuniões com a equipe e uma troca mais eficiente

de informações. Já no trabalho remoto, foi possível realizar as mesmas atividades, mesmo com o distanciamento, porém foi percebido uma falta da troca de conhecimento presencial para melhor aprendizado e fixação.

REFERÊNCIAS BIBLIOGRÁFICAS

KOPPALA, J. **Erp solution with reactjs**. Metropolia Ammattikorkeakoulu, 2018.

GERDESSEN, Anton. **Framework comparison method: Comparing two frameworks based on technical domains, focussing on customisability and modifiability**. Master's thesis. UvA: University of Amsterdam, 2007.

JOHNSON, Ralph E. **Components, frameworks, patterns.** in: *Proceedings of the 1997 Symposium on Software Reusability*, SSR '97, New York, NY, USA, 1997. p. 10–17.

ROBBINS, Jennifer Niederst. **Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics**. Cambridge, UK: O'Reilly Media, 2012.

SEVERANCE, Charles. **JavaScript: Designing a Language in 10 Days.** 8p. Information Technology – University of Michigan, Michigan, 2012.

JR Lewis, M Moscovitz. **CSS Avançado**. Novatec. Tradução de Edgard B, 2010, p33. Disponível em: <https://s3.novatec.com.br/capitulos/capitulo-9788575222201.pdf>. Acessado em Junho 2020.

FACEBOOK, React DOM Documentation 2020. Disponível em: <https://reactjs.org/docs/getting-started.html>. Acessado em junho de 2020.

SCOTTI, Guilherme. **ANÁLISE COMPARATIVA DE FRONT-ENDS DE FRAMEWORKS BASEADOS EM JAVASCRIPT**. 2019. Disponivel em: <https://repositorio.ufmg.br/bitstream/1843/32038/1/GuilhermeScotti.pdf>. Acessado em junho de 2020.

REACT. *React - A JavaScript library for building user interfaces*. 2020. Disponível em: <https://reactjs.org/>. Acessado em junho de 2020.

KHUAT, T. **Developing a frontend application using ReactJS and Redux**.

Dissertação (Degree Programme in Business Information Technology Bachelor's) — Laurea University of Applied Sciences, Leppävaara, 2018. Disponível em: https://www.theseus.fi/bitstream/handle/10024/150837/Tung_Khuat_1301747_Thesis.pdf?sequence=1. Acesso em: 12 jun. 2020.

BANKS, A.; PORCELLO, E**.** ***Learning React: Functional Web Development with React and Redux***. [S.l.]: "O'Reilly Media, Inc.", 2017. Acessado em junho de 2020.

WOOD, L. et al. **Document object model (dom) level 1 specification**. *W3C recommendation*, v. 1, 1998. Acessado em junho de 2020.

AHO, Jenni. Kotisivujen toteutus Lumica Creativelle React-kirjastoa käyttäen. 2020. Disponivel em: <https://www.theseus.fi/bitstream/handle/10024/333517/JenniAhoOpinnaytetyo.pdf?sequence=2>. Acessado em junho de 2020.

MARTTILA, Riku . **HANDLING UNIDIRECTIONAL DATA FLOW IN A REACT.JS APPLICATION**. 04 de maio de 2016. 44p. Tese - Tampere University Of Technology. FONSECA, C. H. G.; BALBINO, F. C. Uma Aplicação Web para Produção de Textos Narrativos com Enredos Alternativos. VIII Congresso Brasileiro de Informática na Educação (CBIE 2019). Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso do Sul Rod. MS-473, km 23, s/n – 79.750-000 – Nova Andradina – MS – Brasil. Disponível em: <https://br-ie.org/pub/index.php/wcbie/article/view/9086/6630> Acessado em junho 2020.

LINS, Gabriel de Souza **- UTILIZANDO REACTJS PARA O DESENVOLVIMENTO DE UM SISTEMA DE ALOCAÇÃO E RESERVA DE SALAS NO CAMPUS DA UFC EM QUIXADÁ** (2019). Acessado em junho de 2020

RIBEIRO, S. **Como Criar uma Aplicação Full-Stack com React**. 2019. Acessado em: Junho de 2020. Disponivel em: <https://medium.com/trainingcenter/construindo-uma-aplicação-full-stack-com-react-a8cab03f0da2> Acessado em julho 2020.

SILVA, F. G. **Uma análise das metodologias Ageis fdd e scrum sob a perspectiva**

do modelo de qualidade mps.br. *SCIENTIA PLENA*, v. 5, n. 12, 2009