FÁBIO JÚNIO ROLIN DE OLIVEIRA

DESENVOLVIMENTO DO MÓDULO DE EVENTOS PARA A PLATAFORMA BLESSS

LAVRAS – MG

2022

FÁBIO JÚNIO ROLIN DE OLIVEIRA

DESENVOLVIMENTO DO MÓDULO DE EVENTOS PARA A PLATAFORMA BLESSS

Trabalho de Conclusao de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

Prof. Dr. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG

2022

FÁBIO JÚNIO ROLIN DE OLIVEIRA

DESENVOLVIMENTO DO MÓDULO DE EVENTOS PARA A PLATAFORMA BLESSS

Trabalho de Conclusao de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

APROVADA em 17 de Agosto de 2022.

Prof. Dr. Paulo Afonso Parreira Junior	UFLA
Bel. Thiago Alves Brum	Zeester

Prof. Dr. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG
2022

À minha mãe, meu pai, minha namorada, meus irmãos e todos meus familiares que sempre acreditaram em mim, fator que nunca me fez desistir perante à nenhuma dificuldade ofertada pela vida.

AGRADECIMENTOS

Agradeço primeiramente a Deus criador de todas as coisas, pois sem ele eu não sou nada e jamais teria chegado até aqui.

Agradeço também a minha mãe Sandra, pessoa mais que especial para mim, que sempre esteve comigo em todos os momentos da minha vida, oferecendo apoio e tudo que eu precisava para ser feliz. Também não posso esquecer do meu pai Antônio, que também é mais que especial para mim e sempre quis me ver vencer todas as barreiras.

À minha namorada Jhessy Maria, que em todo o momento da minha graduação esteve presente, me oferecendo apoio, amor e carinho, além de muita paciência. Te agradeço por tudo e sempre quero estar ao seu lado.

Agradeço a meus irmãos Marcos, Sandy e Juliano, por compartilhar comigo todos os melhores momentos da minha vida, sem vocês nada disso seria possível.

Aos meus tios paternos, agradeço por tudo que sempre fizeram por mim, me ajudando, motivando, e sempre cuidando de mim em todos os momentos necessários.

Também sou grato por ter os colegas do apartamento 304 do brejão, que sempre estiveram comigo durante todo o meu processo de formação.

Também agradeço meus colegas de equipe na Zeester e amigos Eduardo, Guilherme, João Pedro e Kaio, e à toda a equipe da Zeester, que me acompanharam durante todo o processo.

Agradeço também ao meu orientador Prof. Dr. Maurício Ronny de Almeida Souza, que me deu a luz de todo o processo de escrita.

Não posso esquecer de agradecer ao Thiago por ter me dado a oportunidade de sempre querer ser melhor a cada dia mais na profissão.

RESUMO

O Blesss é uma plataforma criada e mantida pela empresa Zeester, voltada para o *streaming* de conteúdo cristão. Neste contexto, a plataforma Blesss está diretamente ligada com a Consciência Cristã, atualmente o maior evento cristão presencial do Brasil. Com a chegada da pandemia de Covid-19, as entidades governamentais foram obrigadas a impor medidas restritivas relacionadas com a aglomeração de pessoas em um mesmo local, fator que tornou impossível a realização da Consciência Cristã de maneira presencial. Logo, surgiu a necessidade do desenvolvimento de uma estrutura extensiva na plataforma Blesss para permitir a realização do evento de forma online. Portanto, o objetivo deste trabalho é descrever as atividades de estágio desenvolvidas na empresa Zeester, com intuito deexecutar o planejamento e a implementação da página para suprir a estrutura de Eventos na plataforma Blesss, a fim de permitir a realização do evento da Consciência Cristã de maneira remota. As principais ferramentas utilizadas na implementação do módulo de eventos na plataforma Blesss foram React JS, Next.js, material UI, Socket.IO, entre outras. O período do estágio realizado pelo autor foi de 07/12/2020 a 07/08/2021. Como resultados, o estagiário desenvolveu o módulo necessário para que fosse possível a realização da Consciência Cristã de maneira online, além de adquirir diversas habilidades voltadas para entendimento de regras de negócios, juntamente com várias habilidades relacionadas com o desenvolvimento Web, principalmente na parte do Front-end das aplicações.

Palavras-chave: Relatorio de estágio. Desenvolvimento Web. Front-end

ABSTRACT

Blesss is a platform dedicated to streaming Christian content created and maintained by the company Zeester. In this context, the Blesss platform is directly linked to “Consciência Cristã”, currently the largest face-to-face Christian event in Brazil. With the arrival of the covid 19 pandemic, government entities were forced to impose restrictive measures related to the agglomeration of people in the same place, a factor that made it impossible to carry out “Consciência Cristã” in person. Through this factor, the need arose to develop a module structure on the Blesss platform to allow the event to be held online. In this sense, the objective of this work is to describe the internship activities developed in the company Zeester, whose objective was the planning and implementation of the page to supply the structure of Events on the Blesss platform in order to allow the realization of the event of the “Consciência Cristã” of remote way. The main tools used to implement the event module on the Blesss platform were React JS, Next.js, material UI, Socket.IO, among others. The period of the internship carried out by the author was from 12/07/2020 to 08/07/2021. As a result, the intern developed the necessary module to make it possible to carry out “Consciência Cristã” online, in addition to acquiring several skills aimed at understanding business rules, along with several skills related to Web development, especially in the Front-end of applications.

Keywords: Internship report. Web development. Front end

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Blesss[1] é uma plataforma voltada para o *streaming* de conteúdo cristão criado e mantido pela empresa Zeester[2]. A plataforma disponibiliza diversos tipos de conteúdos, sendo eles artigos, pregações, séries, vídeos e livros. Dentre os diversos conteúdos apresentados, a pandemia de COVID-19 forçou com que a realização de eventos presenciais se tornasse remota. Por isto, surgiu a necessidade do desenvolvimento de uma estrutura extensiva na plataforma Blesss para permitir a realização de eventos online.

Neste contexto, o objetivo deste documento é descrever as atividades desenvolvidas pelo autor durante o estágio supervisionado realizado na empresa Zeester. O objetivo do estágio foi o desenvolvimento do módulo de eventos na versão Web da plataforma Blesss. O estágio foi realizado no período de 07/2020 à 08/2021, totalizando 990 horas.

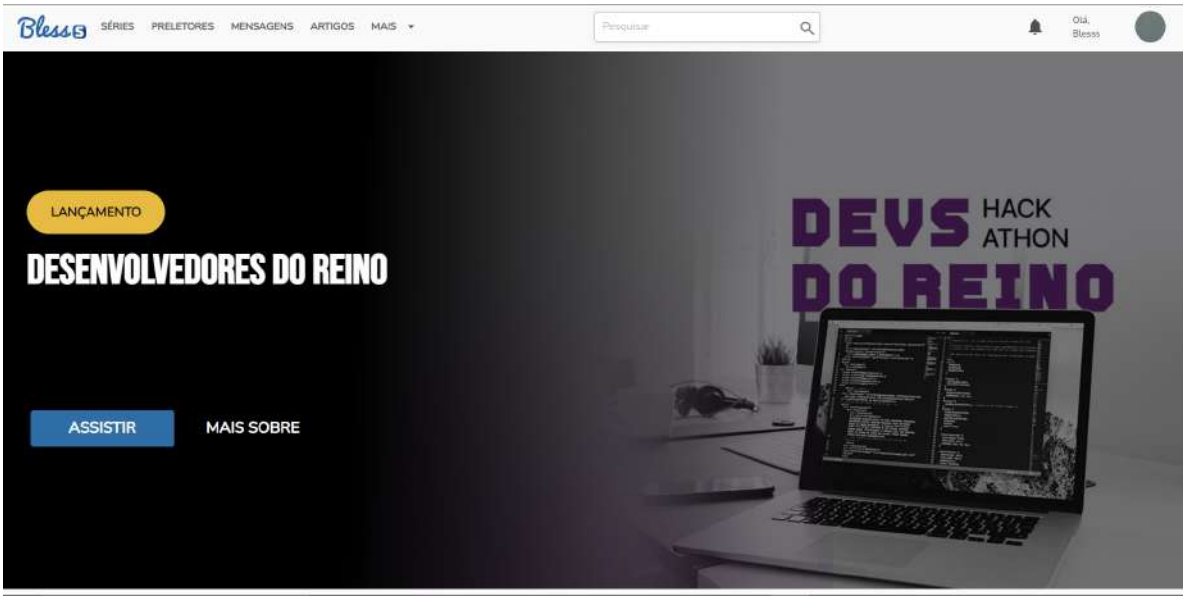
1.1 Sobre a Zeester

A empresa Zeester é uma empresa de pequeno porte do ramo de desenvolvimento de *software*, fundada no ano de 2015 e situada na cidade de Lavras. A empresa é focada em diversas áreas da tecnologia, dentre elas, criação e desenvolvimento de negócios, marketing estratégico e de ação, capacitação de equipes e liderança, marketing, treinamento, consultoria para *startups* e negócios locais, design de interação e usabilidade e desenvolvimento Web fazem parte do seu ecossistema.

A plataforma Blesss é o principal produto criado e mantido pela Zeester. A plataforma conta com versões Web, Mobile (android e iOS) e também para Smart TV's (com sistema Android). A plataforma Blesss oferece a maior parte dos conteúdos de modo gratuito para seus usuários, porém, conta com assinatura que provê conteúdos exclusivos e inéditos para aproximadamente 1700 assinantes. A Figura 1.1 apresenta a página inicial da plataforma, na qual é possível ver os conteúdos existentes na mesma, além de um anúncio de uma série.

[1] https://blesss.org/
[2] https://zeester.com.br/

Figura 1.1 – Página inicial da plataforma Blesss



Fonte: (Blesss, 2022)

Atualmente a empresa conta com cerca de 15 funcionários que são divididos entre as equipes de liderança, *design*, *marketing*, comunicação e desenvolvimento. A principal metodologia ágil utilizada na empresa é o Scrum, que é aplicado em todos os processos relacionados aos produtos e serviços desenvolvidos pela entidade. Durante o estágio, o estagiário participou da equipe de desenvolvimento, responsável por manter a plataforma no ar, além de desenvolver todas as novas funcionalidades projetadas para a mesma.

1.2 Organização do Trabalho

Além deste capítulo introdutório, este documento está organizado nos seguintes capítulos. O **Capítulo 2** descreve os conceitos e tecnologias aplicados no desenvolvimento do software. No **Capítulo 3**, são descritas as atividades de planejamento, projeto e implementação da extensão de eventos na plataforma Blesss. O **Capítulo 4** aponta as principais considerações do autor sobre a experiência obtida com o estágio realizado.

2 CONCEITOS E TECNOLOGIAS

Neste capítulo são apresentados os principais conceitos e definições relacionados às tecnologias utilizadas no projeto desenvolvido durante o estágio realizado na empresa Zeester. A Seção 2.1 trata de assuntos relacionados a programação Web, e também, acerca das tecnologias presentes no ecossistema de desenvolvimento Web que foram essenciais para a resolução dos problemas coexistentes. A Seção 2.2 aborda a metodologia de desenvolvimento adotada para que fosse possível realizar todas as atividades desenvolvida durante o período do estágio em questão. Por fim, a Seção 2.3 discute sobre controle de versão juntamente com o sistema utilizado para fazer a gerncia dos arquivos de todos os componentes criados no projeto.

2.1 Desenvolvimento Web

O desenvolvimento Web compreende todos os aspetos do desenvolvimento de um site ou de uma aplicação para a Web (PORTELA; QUEIRÓS, 2018).

Coulouris et al. (2013) define que o desenvolvimento Web é dividido em três principais pilares:

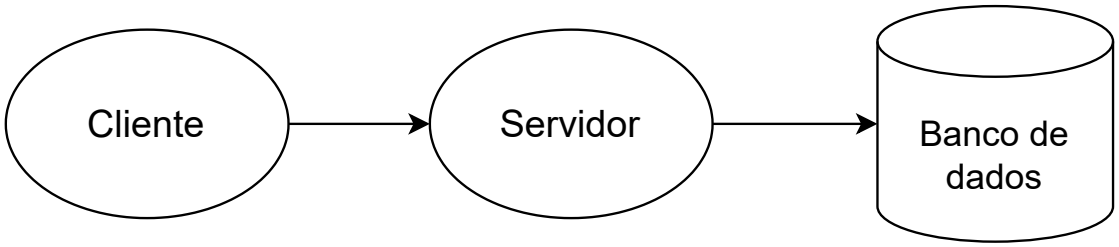
- **Interface**: É a composição de toda a parte visual do sistema e que seja passível de interação;
- **Servidor**: Possui a responsabilidade de executar todo o processamento dos dados gerados pelas interfaces e também, armazena toda a regra de negócio da aplicação;
- **Base de dados**: Armazena todos os dados necessário que foram gerados a partir do processamento feito pelos servidores;

Uma arquitetura que está diretamente vinculado com os pilares apontadas pela definição do desenvolvimento Web é o modelo cliente-servidor. Este modelo é uma arquitetura de software composta tanto pelo cliente quanto pelo servidor, onde os clientes sempre enviam solicitações enquanto o servidor responde às solicitações enviadas pelo cliente. Cliente-servidor fornece uma comunicação entre processos, uma vez que envolve a troca de dados do cliente e do servidor em que cada um deles desempenha funções diferentes (OLUWATOSIN, 2014). A combinação dessas duas partes atuam de modo que o servidor é composto por qualquer máquina que detém uma cópia completa de um ou mais

bancos de dados, já um cliente é capaz de acessar dados presentes em qualquer servidor com o qual possa se comunicar (JING; HELAL; ELMAGARMID, 1999).

A comunicação realizada entre cliente e servidor é possível por meio do protocolo HTTP [1] (Hypertext Transfer Protocol). Na Figura 2.1 é apresentado um diagrama do funcionamento do modelo cliente-servidor, onde o cliente realiza solicitações ao servidor, que por sua vez consulta os dados no banco de dados, e esses dados são retornados para o cliente. É válido ressaltar que o diagrama da Figura 2.1 apresenta a comunicação apenas de um cliente com um servidor.

Figura 2.1 – Diagrama do modelo cliente-servidor



Fonte: do autor, adaptada de (OLUWATOSIN, 2014)

O funcionamento do modelo cliente-servidor aplicado ao ambiente Web é comumente conhecido através dos termos *Back-end*, referente às responsabilidades do servidor, e *Front-end*, referente às responsabilidades do cliente. Para uma melhor compreensão dos termos previamente abordados, as subseções 2.1.1 e 2.1.2 dispõem de explicações mais aprofundadas.

2.1.1 *Back-end*

Dresher, Zuker e Friedman (2018) definem *Back-end* como o lado do sistema que uma ou mais aplicações estão rodando no servidor, em que o propósito é responder a requisições de outras aplicações. O *Back-end* proporciona a criação e a definição da estrutura que rege a regra de negócio (armazenamento e processamento de dados por trás de uma aplicação). Logo, é responsável pelo

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

processamento da interação dos usuários realizadas no cliente, validação dos dados de entrada, e segurança da informação, que é importante nessa comunicação por parte de sua infraestrutura. Na construção de um *Back-end* podem ser utilizadas uma diversidade de linguagens de programação. PHP [2], Java [3], Ruby [4], JavaScript [5] e Python [6] são alguns exemplos de linguagens que podem ser utilizadas nesse processo.

Atualmente, o *Back-end* é comumente desenvolvido no formato de API (Application Programming Interface). As API são interfaces utilizadas no desenvolvimento de sistemas que se comunicam com bibliotecas ou qualquer outros programas (MURPHY et al., 2018). As API são construídas para prover uma padronização na comunicação com outras aplicações, para garantir o encapsulamento de regras de negócio, e diminuir acoplamento, não sendo necessário alterar seu funcionamento para uso por diferentes aplicações. Algumas ferramentas para a construção de API são derivadas das linguagens mencionadas acima, Codeigniter [7], Spring Boot [8], Ruby on Rails [9], Node.js [10] e Django [11] respectivamente, são bons exemplos desses *frameworks*.

As subseções seguintes descrevem o padrão REST, um formato comumente utilizado na criação de API's (Seção 2.1.1.1), e tecnologias de *Back-end* utilizadas no projeto descrito neste documento: a linguagem JavaScript e o *framework* Node.js.

2.1.1.1 REST

O padrão REST é um estilo arquitetural de software para criar serviços Web e auxiliar na integração de sistemas, utiliza o protocolo HTTP para criar serviços que retornam dados geralmente nos formatos XML ou JSON (*JavaScript Object Notation*) (LECHETA, 2015). Assim, é um padrão que

[2] https://www.php.net/
[3] https://www.w3schools.com/java/java$_{i}ntro.aspRuby$
[4] https://www.ruby-lang.org/en/
[5] https://www.w3schools.com/js/
[6] https://www.python.org/
[7] https://www.codeigniter.com/
[8] https://spring.io/
[9] https://rubyonrails.org/
[10] https://nodejs.org/en/
[11] https://www.djangoproject.com/

procura amenizar problemas recorrentes na comunicação entre o *Front-end* e o *Back-end* de aplicações Web.

O termo "RESTful" foi criado para indicar que determinado sistema ou serviço segue os princípios do REST (LECHETA, 2015). A aderência ao padrão RESTful garante reusabilidade às API da aplicação, podendo ser consumidas por aplicações Web, *mobile* e também *desktop*. Essa característica é importante, pois os dados utilizados por essas aplicações de diferentes plataformas serão concisos e sincronizados, evitando a inconsistência nos dados consumidos, além de facilitar na integração com sistemas que não necessariamente fazem parte de uma mesma organização.

O protocolo HTTP oferece alguns verbos que são utilizados na comunicação entre clientes e servidores, para que a mesma possa ser padronizada. Os verbos GET, POST, PUT e DELETE respectivamente possuem as responsabilidades de obter, inserir, atualizar e deletar as informações presentes na base de dados (BERNERS-LEE; FIELDING; FRYSTYK, 1996). Os *endpoints* podem ser compreendidos como endereços existentes nos servidores para que os clientes possam acessá-los e realizar as ações baseadas nos verbos HTTP. A união da combinação dos verbos HTTP e *endpoints* fundamentam a ideia por trás da construção de APIs.

De acordo Lecheta (2015), o padrão REST oferece alguns modelos que são utilizados por parte de toda a estruturação das APIs. No padrão REST, existem alguns modelos semânticos que seguem a padronização oferecida pelos verbos HTTP, sendo eles específicos para cada tipo de interação que rege entre o cliente e o servidor, tais modelos podem ser melhor compreendidos a seguir:

- POST: é utilizado para inserir dados.
- GET: é utilizado para listar dados;
- PUT: é utilizado para atualizar dados;
- DELETE: é utilizado para apagar dados.

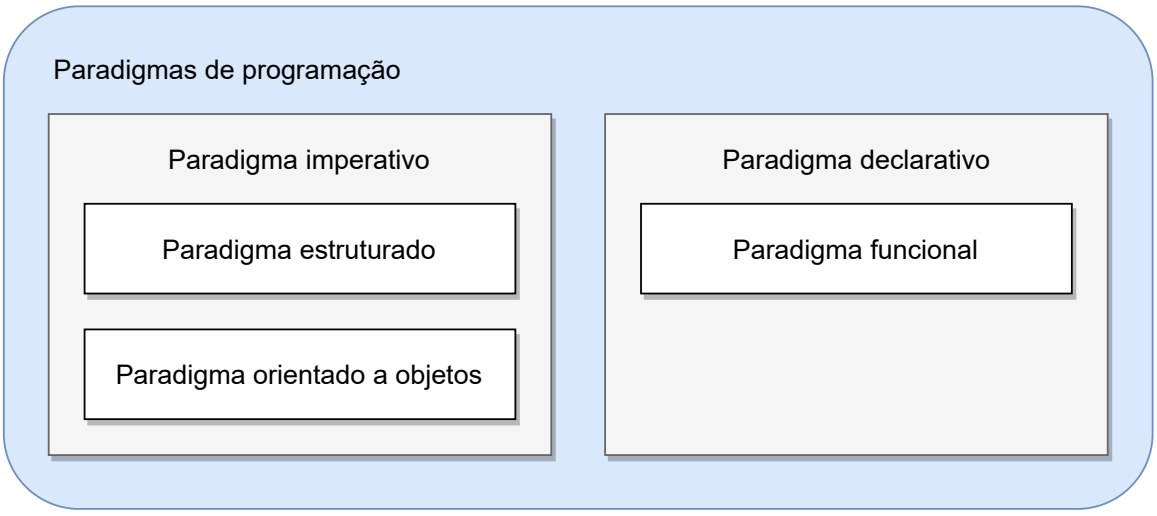
2.1.1.2 JavaScript

JavaScript é uma linguagem de programação inicialmente criada para a Web, sendo utilizada em cerca de 97% dos sites (Q-SUCCESS, 2022). A ampla maioria dos sites modernos utilizam Ja-

vaScript, e todos os sites modernos acessados através de computadores de mesa, consoles de jogos, tablets e smartphones, incluem interpretadores JavaScript. Como consequência, é uma linguagem suportada por todos os navegadores Web existentes (CANIUSE, 2022). O JavaScript é uma linguagem de alto nível, dinâmica, interpretada e não tipada, conveniente para estilos de programação orientados a objetos e funcionais (FLANAGAN, 2004).

Por mais que o JavaScript seja uma linguagem desenvolvida para a Web no *Front-end* (Termo definido na Subseção 2.1.2) e *Back-end*, existem linguagens que oferecem alternativas para o cliente desenvolvido em JavaScript. Python e Dark, por exemplo, são linguagens que oferecem soluções para o desenvolvimento *Front-end* através do Pyscript e Flutter, respectivamente. Contudo, estes *frameworks* não são nativamente interpretados pelo browsers, e dispõem da necessidade de compiladores para seu funcionamento. Um fator que leva o JavaScript a ser uma linguagem amplamente utilizada, está na sua característica multiparadigma, aspecto que proporciona uma série de diversidades durante o seu uso no desenvolvimento, tais disparidades podem ser observadas através da Figura 2.2.

Figura 2.2 – Paradigmas de programação do JavaScript



Fonte: do autor

A partir do diagrama apresentado na Figura 2.2, é possível notar que a linguagem oferece dois paradigmas principais que se ramificam em outros paradigmas, o paradigma imperativo e o paradigma declarativo.

- **Paradigma Imperativo**: Combina um conjunto sequencial, explícito e bem definido de passos que buscam alterar o estado atual do programa a partir de variáveis armazenadas em memória, cujo objetivo é chegar em um resultado final esperado;
- **Paradigma Declarativo**: Determina que as instruções informadas ao computador especifiquem sua determinada lógica juntamente com o seu resultado esperado, não sendo necessário que os fluxos de controle estejam explicitados;

As ramificações presentes no paradigma imperativo são comuns em outras linguagens de programação, e também estão presentes no JavaScript. Os paradigmas em questão são:

- **Paradigma Estruturado**: Refere-se ao paradigma de programação que possibilita a criação de programas utilizando estruturas condicionais (IF/ELSE, SWITCH com CASE), loops (FOR, WHILE, recursividade) e também chamadas de subrotinas;
- **Paradigma Orientado a Objetos**: Busca determinar que a modelagem do código criado seja próximo da realidade, que o mesmo seja composto por uma estrutura formada por objetos, ações específicas, características determinadas, sendo possível a comunicação entre os objetos existentes;

Assim como apresentado na Figura 2.2, o paradigma declarativo também é presente na linguagem JavaScript e é ramificado em um paradigma muito utilizado na linguagem, o paradigma Funcional.

- **Paradigma Funcional**: Foge da ideia do paradigma imperativo, de modo que seu funcionamento é fornecer resultados esperados sem ter a necessidade de especificar o passo a passo para chegar nestes resultados.

Além disso, o JavaScript é uma linguagem fracamente tipada, não sendo necessário especificar tipos quando as variáveis são declaradas. Na Figura 2.3 estão presentes alguns exemplos da linguagem seguindo os paradigmas explicitados anteriormente.

Figura 2.3 – Exemplo de trecho de código da linguagem JavaScript

```
// Exemplos da aplica  o dos paradigmas no Javascript

// Paradigmas em Javascript
array = [4, 2, 10, 11, 5, 2]; //Um array com valores predefinidos

//Exemplo com o paradigma Imperativo
for (let i = 0; i < array.length; i++) {
    // Impress o dos valores do array 4, 2, 10, 11, 5, 2
    console.log(array[i]);
}

// O mesmo exemplo apresentado acima com o paradigma Funcional
// Impress o dos valores do array 4, 2, 10, 11, 5, 2
array.forEach((value) => console.log(value));
//Exemplo com a p l i c a o estruturado utilizando estruturas condicionais
for (let i = 0; i < array.length; i++) {
    // Imprime os valores pares utilizando o paradigma Imperativo
    if (array[i] % 2 === 0) console.log(array[i]);
}

// Imprime os valores pares utilizando o paradigma Funcional
array.forEach((value) => (value % 2 === 0 ? console.log(value) : null));

// Exemplo com o paradigma Orientado a objetos
// C r i a  o de um objeto Carro
class Car {
    constructor(name, color, doors) {
        this.name = name
        this.color = color
        this.doors = doors
    }
}

const car  = new Car("fusca", "blue", 2)
// Imprime o carro criado
console.log(car)
```

Fonte: do autor

No projeto desenvolvido durante o estágio o JavaScript teve uma grande importância, sendo utilizado como linguagem base para que todas as tecnologias presentes na aplicação final. Tudo isso é possível através do Noje.js, ferramenta que será melhor apresentada e conceituada na Seção seguinte.

2.1.1.3 Node.js

O Node.js [12] é um ambiente *open-source* multi-plataforma criado com o intuito de oferecer suporte para o JavaScript rodar fora dos navegadores de internet (NODE.JS, 2022). A criação do Node.js foi viabilizada pela utilização do motor do google chrome V8, que é definido como sendo o mecanismo JavaScript e WebAssembly de alto desempenho de código aberto do Google, escrito em C++ (V8, 2022). O Node.js é single-thread, logo, sistemas que utilizam Node.js não sofrem de *deadlocks* de memória (PEREIRA, 2014).

O funcionamento do Node.js é baseado em um componente denominado *event-loop*, que é o agente responsável por escutar e emitir eventos no sistema. Na prática, ele é um *loop* infinito que a cada iteração, verifica em sua fila de eventos se um determinado evento foi emitido (PEREIRA, 2014).

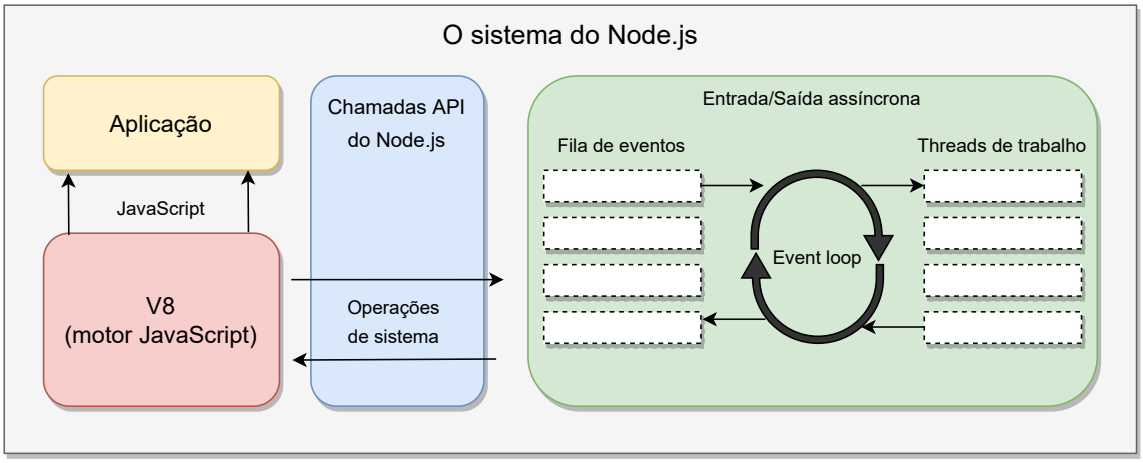
A Figura 2.4 apresenta o diagrama de funcionamento do Node.js, em que seu fundamento se dá sobre a comunicação de uma aplicação com o motor V8 do JavaScript, que é processada por chamadas de entrada e saída na fila de eventos, que por sua vez, realiza as execuções no *event-loop* e retorna a resposta assim que o processamento foi finalizado.

Um fator importante a ser citado, é que o Node.js é altamente escalável, e também, é uma ferramenta fácil de se utilizar, além de ser performático para aplicações que necessitam de uma grande quantidade de entradas e saídas acontecendo durante seu funcionamento, que na maioria das vezes, são programas que geralmente precisam esperar por resultados (também chamadas de aplicações assíncronas), como consultas no banco de dados, feedback de um serviço da Web de terceiros ou solicitações de conexão (BANGARE et al., 2016).

Atualmente o Node.js possui uma grande relevância, uma vez que é base para uma diversidade de ferramentas, o React [13] (melhor definido na Subseção 2.1.2.1) por exemplo, é uma biblioteca que

[12] https://nodejs.org/pt-br/
[13] https://reactjs.org/

Figura 2.4 – Esquema de funcionamento do Node.js



Fonte: do autor, adaptada de VSKILLS (2022)

surgiu através do Node.js. Também é válido ressaltar que o Node.js é uma alternativa bastante utilizada para a construção de API's, fator que colabora para uma comunicação rápida e prática com aplicações *Front-end* pelo fato de que ambos possuem JavaScript como linguagem de programação principal.

2.1.2 *Front-end*

Na Web, a representação do cliente (oriundo do modelo Cliente-servidor) é abordada através do termo *Front-end*, que pode ser melhor compreendido como sendo a parte visual de uma página Web que é passível de interação. Algumas linguagens comumente usadas em conjunto no desenvolvimento do *Front-end* de aplicações são: o HTML [14] (*Hypertext Markup Language*), que é uma linguagem de marcação responsável por estruturar uma página; o CSS [15] (*Cascading Style Sheets*), que define propriedades de estilo da página em questão; e o JavaScript, que adiciona animações e também eventos relacionados com a interação do usuário. Levando em consideração a interação do usuário, algumas métricas são importantes para avaliar se a construção de uma página *Front-end* é adequada a questões de uma boa usabilidade.

[14] https://www.w3schools.com/html/
[15] https://www.w3schools.com/css/

No desenvolvimento do projeto executado durante o estágio na empresa Zeester, foram utilizadas diversas tecnologias para a criação do *Front-end* do sistema em questão. O React JS foi a principal biblioteca utilizada para criar as interfaces da aplicação. O Next.js [16] foi a ferramenta responsável por oferecer suporte a renderizações pelo lado do servidor ao React. O Redux [17] foi utilizado para o gerenciamento de estados e informações globais utilizado na aplicação. A Material UI [18] foi a biblioteca de componentes escolhida para construção das páginas juntamente com o React. Para as chamadas a API, a fim de realizar requisições ao servidor, foi utilizado o Axios [19]. Por fim, foi utilizado o Socket.io [20], tecnologia que é necessária para que o *chat* de mensagens da aplicação funcione com o aspecto de tempo real. Todos os termos e ferramentas mencionadas são descritos nas subseções seguintes.

2.1.2.1 React JS

React JS é uma biblioteca JavaScript para o desenvolvimento de componentes de UI (interface de usuário) reutilizáveis. Atualmente, o React é a biblioteca de *Front-end* mais popular. Foi criado e é mantido pelo Facebook, Instagram e pela comunidade de desenvolvedores individuais juntamente com diversas organizações (AGGARWAL, 2018).

O principal diferencial do React em relação as demais tecnologias utilizadas na construção de interfaces modernas é a Virtual DOM (*Document Object Model*). Uma DOM é uma interface de programação que os navegadores utilizam para representar páginas na Web.

A DOM padrão do HTML busca alterar os componentes existentes na interface percorrendo os *nós* até encontrar o desejado e que será passível de alteração, e através disso, o tempo para encontrar esse *nó* pode ser demorado. Na Virtual DOM, os elementos existentes na interface possuem uma referência direta para sua localização, fator que evita qualquer tipo de busca, melhorando a performance da aplicação, e possibilitando a criação de interfaces mais rápidas e fluidas.

[16] https://nextjs.org/
[17] https://redux.js.org/
[18] https://mui.com/pt/
[19] https://axios-http.com/
[20] https://socket.io/

A base principal do React é formada pelo conjunto de linguagens HTML, CSS e JavaScript. Porém, uma característica importante está na sua finalidade de criar páginas SPA [21] (*Single page aplication*). Diferente dos sistemas Web tradicionais, o usuário navega entre diferentes páginas que são buscadas separadamente do servidor através da solicitação. Em aplicações SPA, há apenas uma página, que contém toda a estrutura da interface, porém, a alteração da página é feita no cliente através do JavaScript (DRESHER; ZUKER; FRIEDMAN, 2018)

A sintaxe do React é chamada de JSX [22], que é semelhante ao HTML e ao XML [23] (*Extensible Markup Language*) (REACT, 2022). Um exemplo básico da sintaxe do React apresentando a criação de um componente pode ser acompanhado abaixo. O componente se trata de um texto seguido de um botão, em que o botão possui o estilo de fundo azul.

[21] https://developer.mozilla.org/en-US/docs/Glossary/SPA
[22] https://reactjs.org/docs/introducing-jsx.html
[23] https://www.w3schools.com/xml/

Figura 2.5 – Exemplo de trecho de código para criação de componente no React JS

```
// Exemplo da criacao de um componente utilizando
// a biblioteca React

import React from 'react';

// Exporta o componente atraves de uma funcao
export default function ReactExample() {
  // Define os estilos visuais das tags JSX
  const style = {
    text : {
      color : "black",
    },
    button : {
      backgroundColor : "blue",
      color : "white",
    },
  };

  // Sintaxe JSX, semelhante ao XML e ao HTML
  return (
    <div className='App'>
      <h1 style ={{ ...style.text }}> Hello World</ h1 >
      <button style ={{ ...style.button }}>This is React !</ button >
    </div>
  );
}
```

Fonte: do autor

Figura 2.6 – Renderização do componente criado na Figura 2.5

Fonte: do autor

2.1.2.2 Next.js

O Next.js é *framework* para React que fornece suporte para criar páginas Web com suporte a geração estática de páginas, renderização de páginas pelo lado do servidor, além de vários recursos que podem ser utilizados durante o desenvolvimento (VERCEL, 2022).

Conforme descrito na subseção anterior, o React constrói aplicações de uma única página (SPA), o que pode ocasionar em problemas relacionados a otimização, uma vez que as aplicações fazem o *download* de todos os arquivos estáticos (HTML, CSS, arquivos JavaScript e imagens) no momento que a requisição é feita, tornando a resposta mais lenta e pesada. Para resolver este problema de otimização, o Next.js oferece ao React a possibilidade de renderizar as páginas Web pelo lado do servidor (*server-side-render*), fator que evita que toda a aplicação *Front-end* seja baixada de uma única vez.

O servidor pode otimizar o tempo de renderização das páginas, uma vez que o mesmo pode selecionar componentes ou até mesmo páginas inteiras e retorná-las quando necessário, não sendo mais preciso que todos os arquivos gerados na construção das páginas sejam retornados para o cliente solicitante (KONSHIN, 2018). Um outro ponto positivo do Next.js está na sua capacidade de oferecer ao React um melhor suporte para o SEO (*Search Engine Optimization*), em que o objetivo é criar estratégias que aumente a posição de *ranking* nos resultados dos mecanismos de busca, melhorando a visibilidade de *sites* criados com o React.

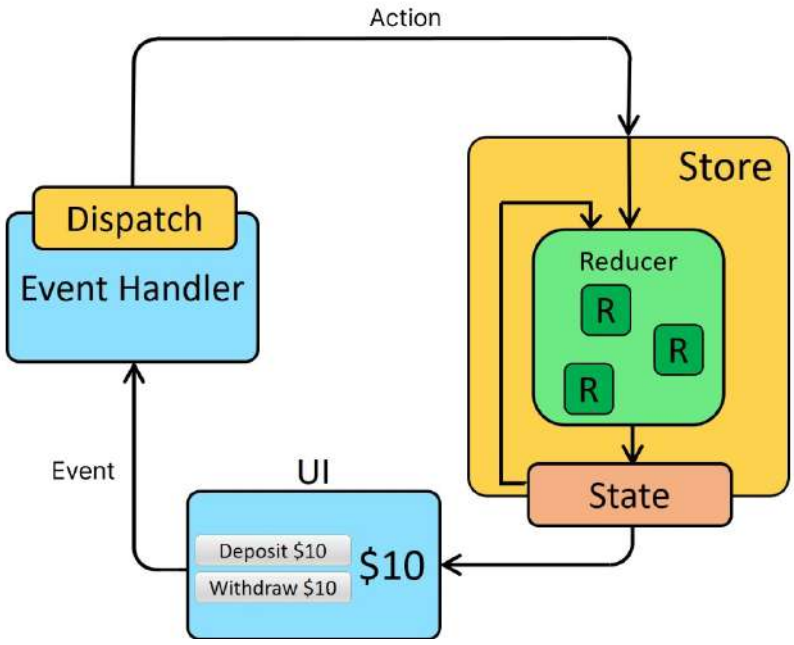
2.1.2.3 Redux

Redux é um padrão e biblioteca para gerenciar e atualizar o estado global da aplicação, usando chamadas de eventos denominadas de *actions* (ações). Ele cria uma *store* [24] da aplicação (serve como um armazenador centralizado de todos os estados que fazem parte da aplicação), com regras que garantem que o estado só possa ser atualizado de maneira previsível (REDUX, 2022).

O Redux é comumente utilizado com o React JS para suprir a necessidade de um gerenciamento global de estado, e basicamente seu fluxo de funcionamento pode ser acompanhado através do diagrama da Figura 2.7.

[24] https://redux.js.org/api/store

Figura 2.7 – Funcionamento do Redux junto com o React



Fonte: do autor, adaptada de REDUX (2022)

No diagrama da Figura 2.7 estão presentes algumas etapas necessárias para a alteração de algum estado global da aplicação, a UI dispara um evento que é captado pelo *Event handler*, que por sua vez da um *dispatch* (disparo) em uma *action*. Logo após o disparo dessa *action*, a mesma é capturada pelo *reducer* (local dentro da store onde é armazenado os estados) e atualiza o estado específico que foi disparado. Após esse fluxo, a informação é atualizada, e como a mesma é consumida pela aplicação a partir dos *reducers* do Redux, todas as interfaces e componentes que fazem o uso do estado são atualizados.

2.1.2.4 Material UI

Material UI é a biblioteca de componentes UI para React JS mais utiliza atualmente, e conta com um conjunto de componentes diversificados prontos para serem utilizados. Atualmente o Material

UI é utilizado por uma diversidade de organizações, Spotify, Amazon, Nasa, Netflix e Unity são exemplos dessas entidades (UI, 2022).

Boduch (2019) define que o Material UI une as duas melhores ferramentas existentes atualmente para a construção do *Front-end*, o React JS e também o material *design* [25], criado pelo Google. Essas características tornam o Material UI atrativo para os desenvolvedores, principalmente pela amplitude e profundidade dos seus recursos.

Figura 2.8 – Exemplo da criação de um botão com o material UI

```
import React from "react";
import Button from "@material-ui/core/Button";

//
function App() {
  return (
    <Button variant="contained" color="primary">
      Button example
    </Button>
  );
}
```

Fonte: do autor

Figura 2.9 – Renderização do componente criado na Figura 2.8

Fonte: do autor

2.1.2.5 Axios

O Axios é um cliente HTTP criado para Node.js e navegadores, que tem como objetivo, oferecer suporte para que seja simples realizar requisições no servidores (AXIOS, 2022). Um fator inte-

[25] https://material.io/

ressante é que o Axios converte as respostas das solicitações para o formato JSON [26], evitando que trabalhos extras sejam necessário.

Por se tratar de uma biblioteca JavaScript, é comum que aplicações *Front-end* fazem o uso do Axios, uma vez que sua integração nessas aplicações é feita de maneira simples. Quando utilizado com o React, o Axios colabora para que a chamada a API's seja feita de maneira simples, fator que colabora para que o padrão REST seja mantido nas requisições com os servidores. O padrão REST é apresentado na Seção 2.1.1.1.

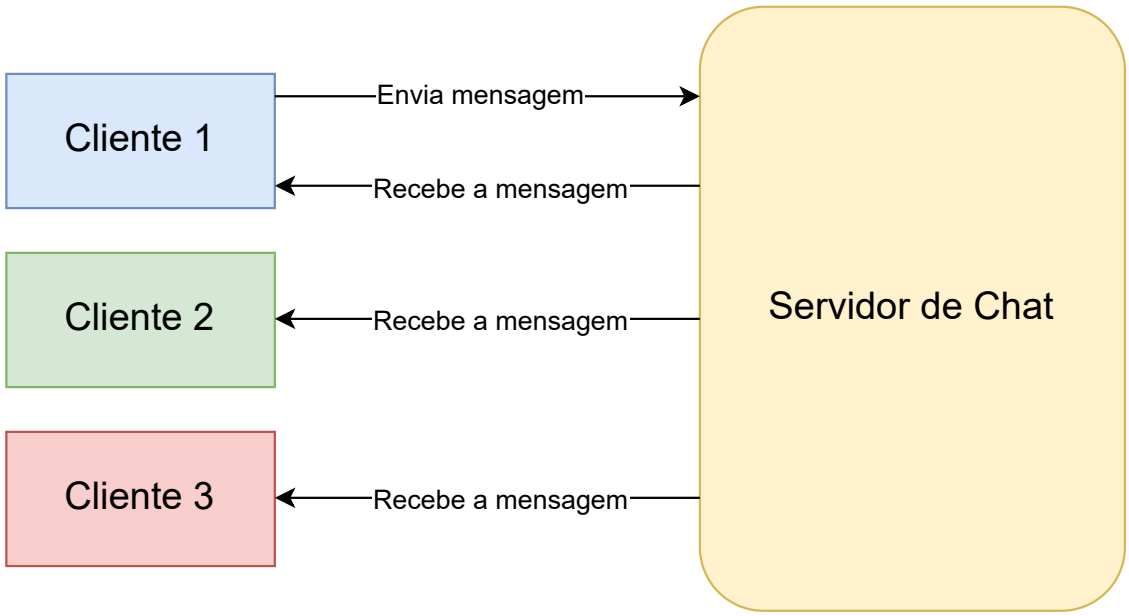
2.1.2.6 Socket.io

Socket.IO é uma biblioteca que permite a comunicação de baixa latência, bidirecional e baseada em eventos entre um cliente e um servidor (SOCKET.IO, 2022). O Socket.IO é uma ferramenta baseada no protocolo de comunicação WebSocket [27], seu principal objetivo é tornar aplicações Web com um aspecto de tempo real. Rai (2013) define aplicações em tempo real como sendo um conjunto de tecnologias e praticas que permitem que usuários recebam informações assim que as mesmas forem publicadas, não sendo necessário que a aplicação cliente fique verificando por atualizações periodicamente.

Considerando o contexto de um *chat* de mensagens de uma aplicação, o funcionamento do Socket.IO pode ser melhor compreendido através da Figura 2.10. A figura representa a comunicação das instâncias de clientes com um servidor de *chat*, em que o Cliente 1 envia uma mensagem e todos clientes (Cliente 1, Cliente 2 e Cliente 3) recebem a mensagem enviada. Esse tipo de ação é denominado como *broadcast*, que notifica todos os usuários que estão conectados em uma rede alguma ação ocorrida.

[26] https://www.json.org/json-en.html
[27] https://en.wikipedia.org/wiki/WebSocket

Figura 2.10 – Aplicação do Socket.IO em um *chat* de mensagens



Fonte: do autor

2.2 Metodologias ágeis

De acordo com Kumar e Bhatia (2012), metodologias ágeis podem ser definidas como um grupo de métodos de desenvolvimento de software que possuem sua base voltadas para modelos iterativos e incrementais. Logo, equipes que seguem metodologias ágeis focam inicialmente na produção do requisitos previsíveis e simples de serem identificados para construir o sistema, e ao longo do fluxo de desenvolvimento, os requisitos vão sendo adicionados de acordo com a necessidade até que a aplicações contenha todas as funcionalidades desejadas. As principais ideias do desenvolvimento ágil de software são introduzidas primeiramente pelo Manifesto Ágil e suas implementações, e depois pela apresentação de práticas ágeis específicas que permitem que as equipes ágeis realizem sua tarefa de desenvolvimento com alta qualidade (HAZZAN; DUBINSKY, 2009).

O Manifesto Ágil foi um movimento cujo objetivo foi levantar padrões relacionados ao processo de desenvolvimento em organizações de desenvolvimento de software distintas. O objetivo do

levantamento destes padrões estavam voltados para a criação de caminhos precisos em relação ao desenvolvimento de software com qualidade e produtividade.

Hazzan e Dubinsky (2009) descreve alguns apontamentos que revolucionou o processo de desenvolvimento de software tradicional. Tais apontamentos são apresentados abaixo:

- **Indivíduos e interações sobre processos e ferramentas**: Os desenvolvedores são encorajados a terem uma preocupação maior com os participantes do processo de desenvolvimento;
- **Software que trabalha sobre uma documentação completa**: A criação de uma documentação antes da etapa de desenvolvimento garante a ordem que os requisitos devem ser cumpridos, além de facilitar para a equipe quais os requisitos que o software deve ter, evitando perda a perda do tempo e mão de obra em relação a um desenvolvimento sem quaisquer tipos de regras e padronizações;
- **Colaboração do cliente na negociação de contratos**: Para um bom desenvolvimento incremental e passivo de alterações com base em seus requisitos, o contato diário do cliente com o processo de desenvolvimento ajuda a evitar problemas ligados ao mal entendimento das necessidades apresentadas pelo mesmo, fator que ajuda a evitar desperdício de recursos e tempo de desenvolvimento;
- **Responder à mudança ao invés de seguir um plano**: Alterações podem surgir durante o desenvolvimento, sendo assim, o fluxo deve ser flexível a qualquer alteração ou requisito que surgir.

A partir destas convicções, as metodologias ágeis de desenvolvimento buscam acoplar um bom planejamento, desenvolver sistemas de forma iterativa, flexível a mudanças, e também, procura priorizar a comunicação do time responsável por utilizá-las. O Scrum é um |*framework para a gestão ágil de projetos* que incorpora todas essas caraterísticas. Na subseção 2.2.1, será melhor explicitado as definições da metodologia em questão, utilizada durante o desenvolvimento da aplicação para cobrir o evento da Consciência cristã 2021.

2.2.1 Scrum

Scrum [28] é um *framework* leve que ajuda pessoas, equipes e organizações a gerar valor por meio de soluções adaptáveis para problemas complexos (RUBIN, 2012). O Scrum é um método iterativo e incremental que busca trabalhar com um *product backlog*, que é uma lista de requisitos/tarefas que define as funcionalidades que o sistema em desenvolvimento deve ter. O foco principal da utilização do *product backlog* é prorizar tarefas que possuem maior importância primeiro, e assim que haver tempo disponível, as tarefas restantes são feitas (Scrum Guide, 2022).

O tempo no modelo Scrum é medido através de uma *sprint*, que é um ciclo de desenvolvimento que pode durar até um mês. Durante uma *sprint*, tarefas com maiores prioridades são selecionadas para serem executadas no período de tempo de duração da *sprint*. No final de uma *sprint*, a equipe de desenvolvimento apresenta o resultado do desenvolvimento realizado no ciclo e verifica se todas as tarefas foram finalizadas, ou se precisam de mais tempo para concluir as que ainda não foram terminadas. Assim que a *sprint* é finalizada, todo o processo é reiniciado com o planejamento da próxima (Scrum Guide, 2022).

Rubin (2012) define que para o funcionamento da metodologia Scrum, a equipe é dividida em algumas entidades que ficam responsáveis por cada parte da aplicação da metodologia, tais divisões são listadas abaixo:

- **Product Owner**: É o dono do produto, responsável por definir a ordem das funcionalidades que serão desenvolvidas, além da prioridade das mesmas.
- **ScrumMaster**: Atua como um líder que procura ajudar todos os membros da equipe envolvidos no projeto a entender os princípios, práticas e valores da metodologia Scrum, além de proporcionar a equipe a sempre melhorar o uso da metodologia Scrum.
- **Development Team**: É a equipe atuante no desenvolvimento das funcionalidades, como designers, desenvolvedores, arquitetos, administradores de bancos de dados e os demais envolvidos;

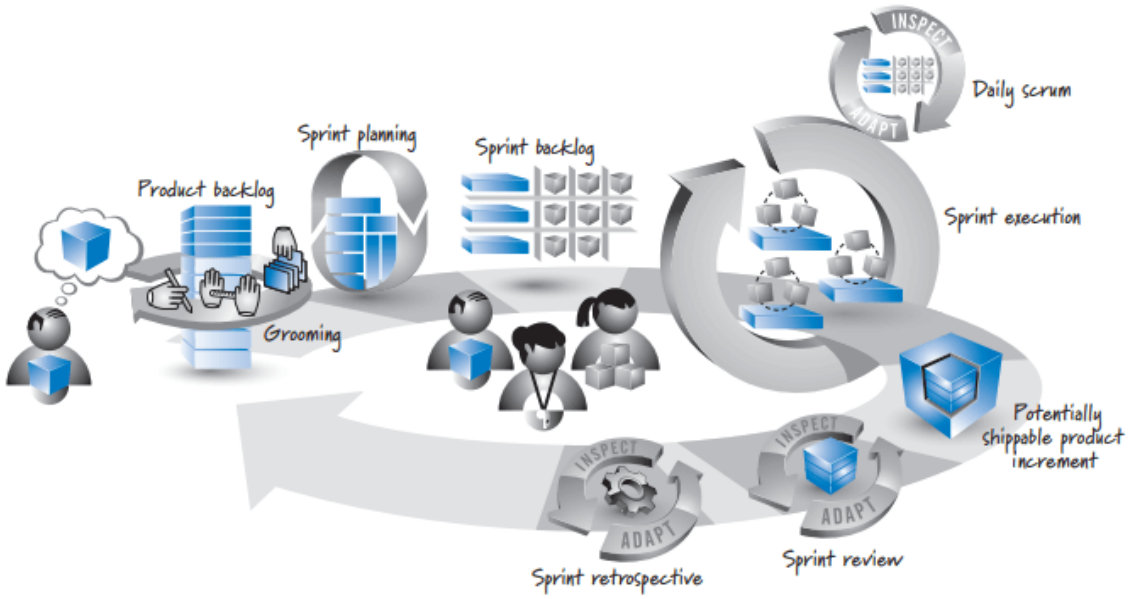
[28] https://www.scrum.org/resources/scrum-guide

Além da *sprint*, o Scrum possui alguns eventos que fazem parte de seu ecossistema e que colabora para todo seu funcionamento a fim de atingir sua proposta. Scrum Guide (2022) define estes eventos como:

- **Sprint Planning**: trata-se da inicialização da *sprint* em questão, fator que estabelece qual será o trabalho que será executado por todos os colaboradores durante o período daquela mesma.
- **Daily Scrum**: são pequenas reuniões diárias que visam inspecionar o progresso em direção a conclusão das demandas, sendo flexível a adaptação das tarefas relacionadas a sprint (*Sprint backlog*) conforme necessário, ajustando o próximo trabalho planejado.
- **Sprint Review**: é o evento acontece no final de uma *sprint*, seu objetivo é inspecionar o resultado da mesma e determinar futuras adaptações. Todo o time presente no modelo Scrum apresenta os resultados gerados de seu trabalho para as principais partes interessadas, a fim de discutir o progresso do produto em questão.
- **Sprint Retrospective**: busca planejar maneiras de aumentar a qualidade e a eficácia das execuções das *sprints*.

Para que seja possível assimilar melhor o fluxo de funcionamento da metodologia Scrum com todas as suas partes, a Figura 2.11 apresenta todas as etapas separadas e ordenadas de acordo com o decorrer de uma *sprint*, de modo que a mesma iniciada, passando por todos os eventos explicitados anteriormente.

Figura 2.11 – Fluxo do funcionamento geral da metodologia ágil Scrum



(RUBIN, 2012)

2.3 Controle de versão

O Controle de versão é um processo responsável por registrar alterações em um ou mais produtos de trabalho ou conjunto ao longo do tempo para que seja possível recuperar versões específicas posteriormente (CHACON; STRAUB, 2014). Desenvolver aplicações em grandes equipes de desenvolvimento é uma tarefa bastante complexa quando o assunto é gerenciamento de versões, pois com base nas equipes e seus afazeres, um mesmo arquivo pode ser completamente distinto nos diversos computadores utilizados pela equipe ao decorrer do desenvolvimento de uma funcionalidade. Assim, os sistemas de controle de versão colaboram para que equipes possam trabalhar sem maiores problemas relacionados com qual versão dos arquivos é a correta, aumentando a produtividade e agilidade nas entregas das demandas.

Apesar de ser uma prática comum utilizar softwares conhecidos como o Git [29], existem maneiras distintas de realizar o controle de versão de um sistema que se encontra em desenvolvimento. Alguns modelos de sistemas para realizar este gerenciamento é listado abaixo.

[29] https://git-scm.com/

- **Sistema de controle de versão local**: Trata-se do armazenamento dos arquivos distintos em um banco de dados local, ou seja, uma base de dados que esteja na máquina do desenvolvedor.

- **Sistema de controle de versão centralizado**: É um modelo que consiste no armazenamento dos arquivos versionados em um servidor, através dessa característica, diversos desenvolvedores podem acessar esse servidor e consultar os arquivos.

- **Sistema de controle de versão distribuído**: Assim como o nome revela, trata-se do armazenamento dos arquivos em diferentes locais, esses locais são as máquinas dos próprios desenvolvedores que detém uma cópia exata dos arquivos presentes no servidor principal. Um fato importante dessa técnica é que caso o servidor pare de funcionar, qualquer um dos desenvolvedores pode simplesmente enviar seu repositório local para o servidor novamente, fator que evita a perda dos arquivos.

Uma aplicação bastante utilizada para fazer o gerenciamento de versões atualmente é o Git[30], que foi o sistema utilizado para fazer o gerenciamento das versões da aplicação desenvolvida para cobrir o evento da Consciência Cristã 2021.

2.3.1 Git

Git é um sistema de controle de versão distribuído que foi projetado desde o início para evitar falhas que existiam em muitos outros sistemas de controle de versão, além de fornecer velocidade, desempenho, flexibilidade e uma boa usabilidade durante seu uso (SOMASUNDARAM, 2013).

Inicialmente o Git foi desenvolvido para ser o sistema que faria controle de todos os arquivos do kernel do Linux [31], e desde então, foi criado para resolver problemas complexos e desafiadores. Alguns fatores que levaram o Git a ser amplamente utilizado é seu foco em aspectos importantes como Atomicidade, Desempenho e também na Segurança. Esse fatores podem ser melhor compreendidos a seguir:

[30] https://git-scm.com/
[31] https://www.linux.org/

- **Atomicidade**: Busca garantir que as suas operações sempre finalizem de maneira segura, evitando que informações se percam no meio do processo.
- **Desempenho**: O Git é uma ferramenta que prioriza o desempenho, uma vez é muito importante que todas suas operações aconteçam de modo mais performático possível.
- **Segurança**: Para manter seguro e integro os arquivos monitorados, o Git utiliza um algoritmo de hash SHA-1, que faz a codificação dos arquivos evitando que os mesmo sejam alterados por terceiros.

As alterações dos arquivos no ecossistema do Git são apresentadas como um *snapshot*, que pode ser entendido como uma maneira de "congelar" os arquivos da maneira que estão quando foram alterados, a logo após este passo, o Git gera um *commit*, que é um pacote onde é armazenado as alterações realizadas, e também, um código identificador que pode ser utilizado para localizar as alterações.

3 DESENVOLVIMENTO DO MÓDULO "EVENTOS" NA PLATAFORMA BLESSS

A Consciência Cristã [1] é um evento tradicional que acontece na cidade de Campina Grande - PB. Sua primeira edição foi em 1999 e atualmente é o maior evento cristão presencial do Brasil. O evento reúne pessoas de diferentes localizações, trazendo assuntos e palestras relacionadas com o cunho cristão no Brasil (VINACC, 2022).

Com a chegada da pandemia de Covid-19, tornou-se impossível a realização do evento na forma presencial, uma vez que a pandemia obrigou as entidades governamentais a impor medidas restritivas relacionadas com a aglomeração de pessoas em um mesmo local (CIOTTI et al., 2020). Dessa forma, o evento só poderia ocorrer na forma online. Com isso, a plataforma Blesss, que é ligada diretamente com a Consciência Cristã (VINACC, 2022), foi alocada para solucionar, por meio da tecnologia o *déficit* ocasionado pela pandemia.

Para solucionar o problema, surgiu a necessidade da criação do módulo denominado de "Eventos" na plataforma Blesss. Blesss é uma plataforma que possui diversos conteúdos cristãos, como séries, artigos, pregações, livros e vídeos. Este novo módulo seria uma extensão da plataforma existente com a missão de oferecer suporte para a realização da Consciência Cristã 2021 online. O foco deste trabalho é apresentar o processo de criação do componente *Front-end* do módulo Eventos da plataforma.

Desta forma, este Capítulo descreve as todas as etapas de desenvolvimento executadas pelo estagiário para a criação do módulo "Eventos" durante o estágio realizado na empresa Zeester. Na Seção 3.1 é apresentado o fluxo de trabalho, juntamente como a metodologia de desenvolvimento adotada pela empresa. Na Seção 3.2 são abordadas as principais concepções da estrutura de Eventos que foi desenvolvido para a plataforma Blesss. A Seção 3.3 descreve o projeto da solução desenvolvida, com o foco na arquitetura do software desenvolvido. A Seção 3.4 apresenta em detalhes todo o procedimento para desenvolver a extensão de eventos da plataforma Blesss, e apresenta o resultado final do produto. Por fim, a Seção 3.6 podera sobre pontos importantes relacionados com a convivência da equipe de trabalho durante o estágio, além de dificuldades e limitações encontradas durante todo o processo de duração do mesmo.

[1] https://conscienciacrista.org.br/

3.1 Fluxo de trabalho na empresa Zeester

A Zeester adota a metodologia ágil de desenvolvimento Scrum, utilizando todas as etapas descritas na Seção 2.2.1. Assim, é importante ressaltar a divisão das equipes presentes na empresa e o fluxo de desenvolvimento com a adoção da metodologia ágil Scrum. A Zeester é organizada nas seguintes equipes:

- **Liderança**: É responsável por tomar todas as decisões relacionadas com os produtos, assim como as definições dos objetivos da empresa como um todo.
- **Design**: Equipe responsável por analisar os requisitos levantados dos produtos a fim realizar prototipações das funcionalidades que deverão ser desenvolvidas.
- **Marketing**: Possui o objetivo de realizar a divulgação da plataforma Blesss, além de divulgar todos os conteúdos que são lançados na mesma.
- **Comunicação**: É responsável por manter a comunicação com todos os usuários da plataforma Blesss para que seja possível sanar os problemas encontrados.
- **Desenvolvimento**: É a equipe destinada a desenvolver todas as funcionalidades apresentadas nos requisitos dos produtos da empresa.

O fluxo de desenvolvimento na empresa é iniciado com o diretor (presente na Liderança) exercendo o papel do *Product Owner*, que por sua vez busca levantar os requisitos por meio da comunicação com o cliente. Uma vez que os requisitos são levantados, o *Product Owner* realiza um processo de refinamento dos requisitos previamente coletados. Em seguida, o *Product Owner* passa os requisitos refinados para a equipe de *design*, para que a mesma possa construir protótipos relacionados com as funcionalidades apresentadas nos requisitos. Uma vez que a equipe de *design* apresenta uma solução, o *Product Owner* realiza reuniões com a equipe de desenvolvimento a fim de debater questões baseadas nos requisitos, e averiguar o nível de complexidade das mesmas. Assim, é possível verificar a disponibilidade da equipe assim como validar a possibilidade de implementação das funcionalidades.

Após o alinhamento com a equipe de desenvolvimento, os requisitos são transformados em tarefas no *Product Backlog*. Na Zeester as *sprints* tem duração de 1 semana, de modo que, no primeiro dia, a equipe de desenvolvimento realiza a *Sprint Review*, a fim de debater questões relacionadas com a última *sprint* e logo em sequência a *Sprint Planning*, que por sua vez, busca definir quais serão as tarefas destinadas para a próxima.

Na *Sprint Review* a equipe de desenvolvimento debate assuntos relacionados com a última *Sprint*, verificando as tarefas que foram concluídas, assim como os impedimentos gerados durante a execução das mesmas. Tais impedimentos são apresentados para que possam ser solucionados a fim de manter o desenvolvimento das tarefas não finalizadas para a próxima *sprint*. É importante ressaltar que a equipe de desenvolvimento também realiza um *Sprint Retrospective*, que busca validar possíveis problemas, a fim de melhorá-los para as próximas *sprints*.

Durante a *Sprint Review*, a equipe de desenvolvimento utiliza a prática denominada *Scrum Poker* ou *Planning Poker*, que é um estilo de dimensionar as tarefas de uma determinada *sprint*, onde todos os desenvolvedores votam a fim de chegarem em um consenso. Os pontos presentes no *Scrum Poker* são chamados de *Story points* (RUBIN, 2012), e na Zeester, as pontuações seguem a sequência de Fibonacci (1, 2, 3, 5, 8, 13, ...). O peso das tarefas votadas pela equipe de desenvolvimento variam de 1 até 5, em que os pesos 1 e 2 remetem a tarefas que são realizadas com o mínimo de esforço e que levam pouco tempo; as tarefas que possuir peso 3, são relativamente mais complexas, e na maioria das vezes levam de 2 a 3 dias para serem finalizadas. Por fim as tarefas de peso 5, são as mais complexas, e na maioria das vezes duram uma *sprint* ou mais. As tarefas avaliadas com valores acima de 5 são quebradas em tarefas menores a fim de amenizar sua complexidade.

Após o final da votação das tarefas, a *sprint* é iniciada com todas as tarefas selecionadas para aquela determinada semana (o *Sprint Backlog*). O fluxo das *sprint* permanece até que a funcionalidade seja entregue pela equipe de desenvolvimento, e a partir deste momento, a funcionalidade desenvolvida é apresentada para o *Product Owner*, que avalia o resultado e verifica possíveis melhorias.

Por fim, assim que o *Product Owner* aprova as funcionalidades desenvolvidas, a equipe de desenvolvimento coloca as mesmas em produção, i.e. disponibiliza todas as novidades para os clientes.

As equipes de *Marketing* e *Comunicação* atuam juntamente com as demais equipes divulgando os lançamentos e também solucionando os possíveis problemas que podem aparecer após as atualizações.

3.2 Concepção do módulo "Eventos" da plataforma Blesss

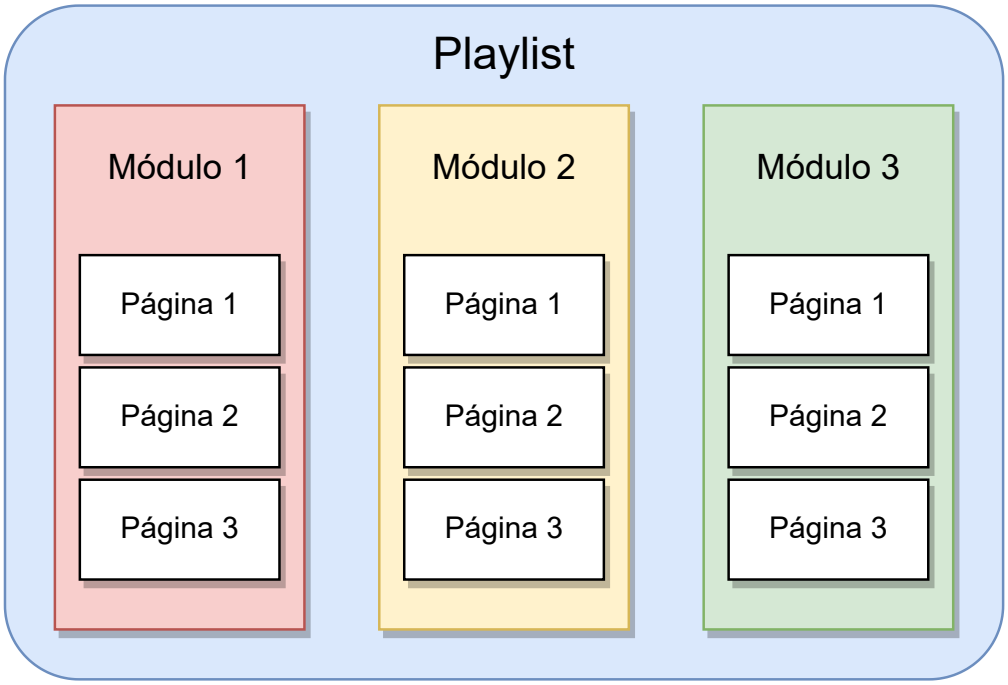
A ideia por trás do módulo de Eventos da plataforma Blesss surgiu com a necessidade da cobertura da Consciência Cristã 2021. Assim, o novo módulo deveria conter alguns aspectos básicos para que fossem passíveis de uso durante a execução do evento.

O planejamento do módulo teve foco em atender o formato presencial, sendo que o evento tinha duração média de 6 dias, e durante os dias de ocorrência do evento, aconteciam diversas palestras com assuntos distintos. As palestras em questão, eram divididas como episódios diários, seguindo a ideia de séries como é conhecido atualmente. A partir deste conceito, o *Product Owner* e as equipes de desenvolvimento e de *Design* propuseram a abstração que atenderia a estrutura do evento em questão:

- **Página**: É a representação da palestra em si. As páginas estão presentes em um Módulo. É na página que as *live*s acontecem por intermédio das palestras ministradas.
- **Módulo**: É um conjunto de Páginas. O Módulo busca representar um assunto específico que irá ocorrer no Evento da Consciência Cristã.
- **Playlist**: Trata-se da representação do Evento em questão, ou seja, a Consciência Cristã. A Playlist armazena os diversos Módulos que serão ministrados durante o evento. É importante ressaltar que a estrutura de Playlist é existente na plataforma para representar assuntos.

A Figura 3.1 representa o escopo da estrutura de Eventos como um todo, sendo possível verificar que uma Playlist possui vários Módulos, assim como um Módulo contém várias Páginas.

Figura 3.1 – Representação da estrutura de Eventos



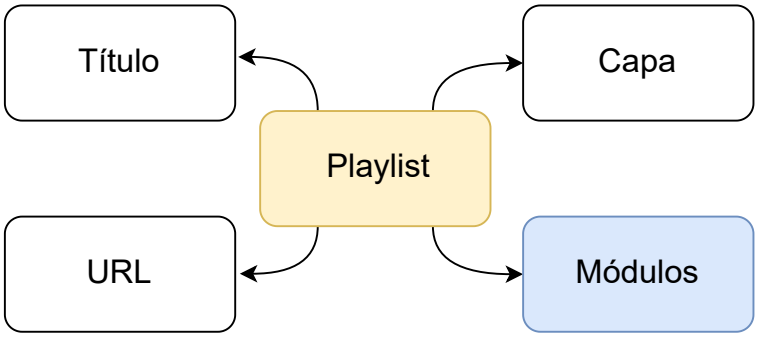
Fonte: do autor

3.2.1 Levantamento de requisitos

Considerando a abstração base da estrutura de Eventos, alguns requisitos foram definidos para detalhar melhor os componentes de *playlist*, *módulo* e *página*.

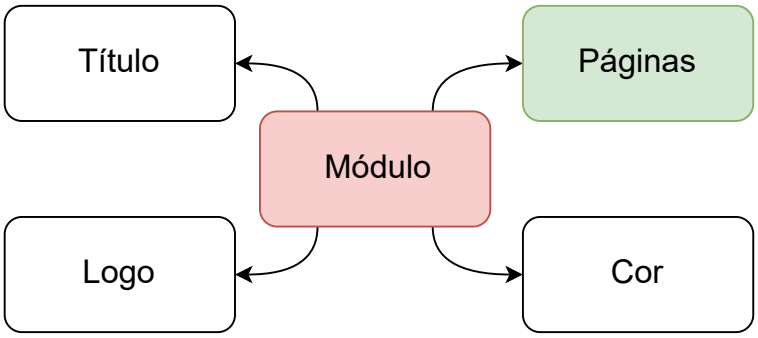
A **Playlist** é a entidade mais externa da representação, e com isso, a mesma conta com uma capa, um título (armazena o nome da mesma), uma URL amigável (utilizada para que fosse possível compartilhar o Evento), e Módulos, que representam os assuntos abordados durante o Evento (Figura 3.2). O **Módulo** por sua vez possui um título, uma cor única utilizada para tematizar as palestras, uma logo, e por fim, as Páginas referentes ao conteúdo ministrado no Módulo (Figura 3.3).

Figura 3.2 – Representação de uma Playlist



Fonte: do autor

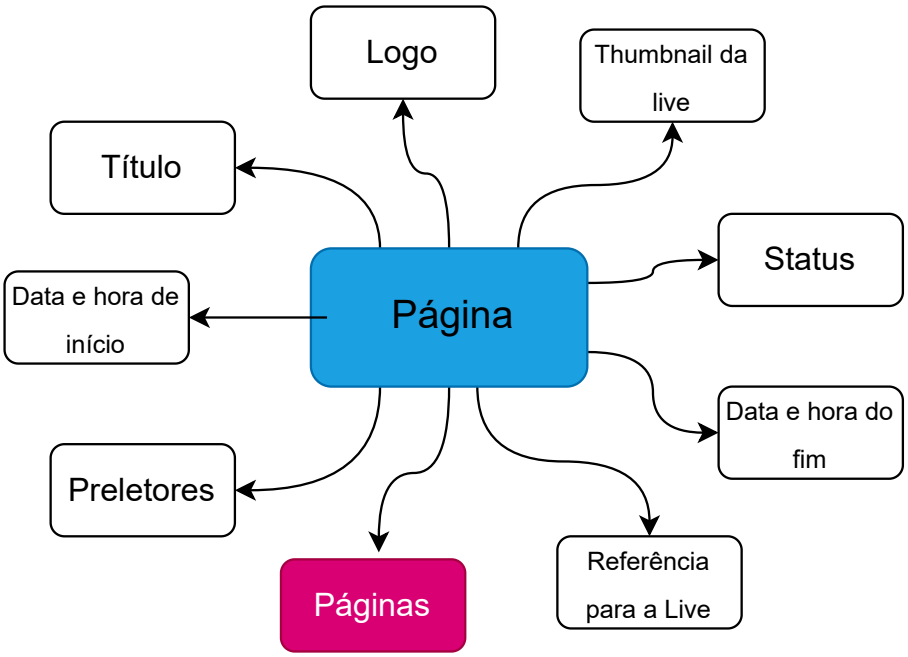
Figura 3.3 – Representação de um Módulo



Fonte: do autor

A **Página** é a entidade mais interna da estrutura de Eventos, e com isso, uma vez que é na mesma que acontece as palestras ministradas durante o evento, ela deve ser composta por um título (armazena o nome do Módulo), uma logo, a data da *live* (início e fim), os preletores que ofertam o conteúdo, uma referncia para o servidor de *streaming* da *live* juntamente com sua *thumbnail*, o *status* que a *live* está (atributo que informa se a *live* já foi iniciada, está em andamento ou foi finalizada), e por fim, as Páginas que fazendo parte do Módulo que serão utilizadas para realizar a navegação entre palestras (Figura 3.4).

Figura 3.4 – Representação de uma Página



Fonte: do autor

A partir dos requisitos definidos e levantados nesta Subseção, é importante ressaltar que este documento tem como foco a construção da representação de Página, e todos as Seções e Subseções a partir da Seção 3.3 focam nos recursos que foram necessários para o planejamento e construção da mesma.

3.3 Projeto da solução

Para a construção da Página presente na estrutura de Eventos, alguns componentes importantes são indispensáveis para que a mesma possa oferecer todos os comportamentos necessários durante as apresentação das palestras presentes na Consciência Cristã. A partir desta afirmativa, esta Seção visa apresentar todo o projeto juntamente com todas as respectivas arquiteturas necessárias para oferecer suporte ao funcionamento da Página como um todo. A Subseção 3.3.1 apresenta todos os componentes da Página, e também, toda a organização dos mesmos na página Front-end resultante.

A Seção 3.3.2 apresenta todos os possíveis estados da Página, sendo apresentado seu comportamento quando a *live* foi iniciada, quando a *live* está *offline*, assim quando a mesma ainda não iniciou. Por fim na Subseção 3.3.3 será apresentado os diagramas da arquitetura necessária para todo o funcionamento da Página.

3.3.1 Estruturação dos componentes da representação de uma Página

Inicialmente é importante ressaltar que para atender uma palestra, alguns componentes são essenciais neste processo. Como a Página é responsável por suportar as palestras ministradas, os seguintes elementos são necessários na composição da mesma, assim como suas respectivas funções:

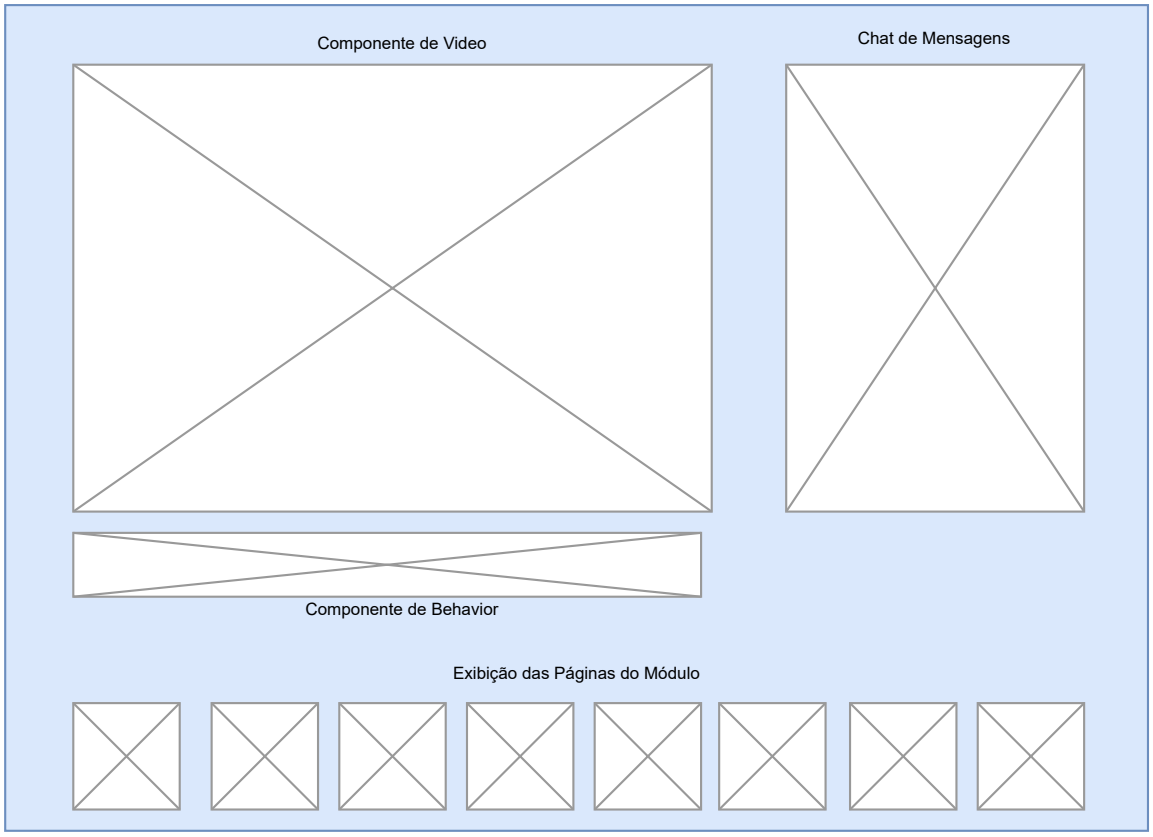
- ***Player* de vídeo**: Responsável pela exibição da *live* da respectiva palestra ministrada.
- ***Chat* de mensagens**: Componente que os usuários poderão trocar mensagens através da interação gerada durante as *live*s.
- ***Behavior***: Trata-se de um componente de interações expressas pelo usuário em que o usuário pode gostar, não gostar, compartilhar e também relatar possíveis problemas presentes na *live* como um todo. Além desta funcionalidade, o componentes também exibe o nome da palestra, assim como todos os preletores presentes na mesma.
- **Exibição das Páginas do Módulo**: Componente que irá mostrar as Páginas presentes no Módulo da *live* que esteja sendo exibida no momento.

Com base no levantamento dos componentes elencados acima, a concepção gerada a partir dos elementos descritos, foi apresentada pela equipe de *design* um protótipo referente a como a página web seria organizada semelhante ao *mockup* da Figura 3.5. É interessante mencionar que a estruturação não fugiu da ideia das plataformas de *streaming* mais famosas. O Youtube [2] por exemplo, possui a estrutura semelhante a concepção gerada. Também é válido citar que a figura é uma representação de baixa fidelidade, cujo o principal objetivo é simular o protótipo apresentado pela equipe de *design*, e

[2] https://www.youtube.com/

com isso, a mesma foi desenvolvida pelo autor que não possui responsabilidade com a área de UI/UX (*User Interface / User Experience*).

Figura 3.5 – Esboço da organização de uma Página



Fonte: do autor

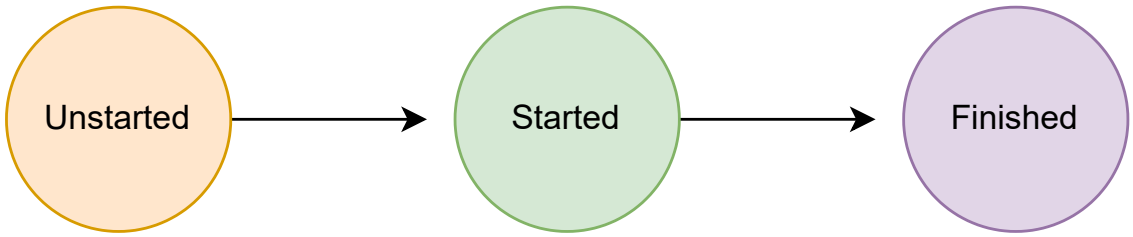
3.3.2 Estados da Página de acordo com o *status* da *live*

Baseado nos atributos apresentados na Subseção 3.2.1, é importante destacar o atributo *status*. Este atributo possui uma grande importância quando o assunto é estado atual da *live* presente em uma Página, uma vez que a mesma possui algumas disposições como por exemplo não iniciada, iniciada e finalizada.

A Página possui três estados principais: “Unstarted”, “Started” e “Finished”. Estes estados são responsáveis por garantir que os componentes presentes na Página reajam de acordo com os valores elencados.

A Figura 3.6 apresenta um diagrama do fluxo dos estados da *live*, sendo possível notar a sequência exata que a Página irá se apresentar.

Figura 3.6 – ciclo de vida da *live* de uma Página



Fonte: do autor

Inicialmente quando o estado é “Unstarted”, significa que o *streaming* ainda não foi inicializado, e com isso, os componentes *player* de vídeo e *chat* de mensagens possui as seguintes configurações:

- ***player* de vídeo**: Exibe a respectiva *thumbnail* existente na Página, de modo que o mesmo apresenta uma *tag* informando ao usuário o dia e o horário de início da palestra que será ministrada.
- ***chat* de mensagens**: O componente não permite que os usuários realizem interações, e com isso, é apresentado textos informando o usuário que a *live* ainda não foi iniciada.

Quando o atributo *status* armazena o valor “Started”, significa que o *streaming* foi inicializado, e com isso, este é o estado que refere ao início da palestra em questão. Os componentes que mudam de estado são os listados abaixo:

- ***player* de vídeo**: Inicia a exibição da *live* em questão, oferecendo controles de pausa/iniciar para o usuário.

- ***chat* de mensagens**: O componente de permite que os usuário iniciem as interações por meio de suas mensagens.

Por fim, o último possível estado do atributo *status* é o "Finished", que faz jus a finalização da *live*. Este estado possui um comportamento diferente dos demais, pois além dos dois componentes que sofrem mudança nos estados anteriores, o componente de exibição das Páginas do Módulo aparece no lugar do *chat* de mensagens. Através desta afirmação, a seguinte configuração é apresentada quando o valor "Finished" é setado:

- ***player* de vídeo**: Volta a exibir a *thumbnail* da Página em questão e finaliza o *streaming* da *live* referente a respectiva palestra.
- ***chat* de mensagens**: É substituído por um componente que exibe as Páginas presentes naquele respectivo Módulo.

È possível notar que em todos os estados o usuário consegue acessar a Página em questão, porém, a mesma irá ter alterações baseada no respectivos estados mencionados, tornando assim o sistema acessível em todas as ocasiões.

Um outro ponto importante a ser levantado é o componente *chat* de mensagens, que possui controles únicos a fim de fazer o controle de todas as mensagens que estão sendo enviadas pelos usuários durante as palestras. Este ponto é importante para que seja viável evitar possíveis mensagens ofensivas, uma vez que a estrutura de Eventos da plataforma Blesss é aberta para todas pessoas que queiram assistir ao evento da Consciência Cristã. Os controles previamente abordados trata-se da possibilidade de bloquear usuários que enviarem mensagens não permitidas por meio de um sistema administrador que será melhor apresentado na Subseção 3.3.3.

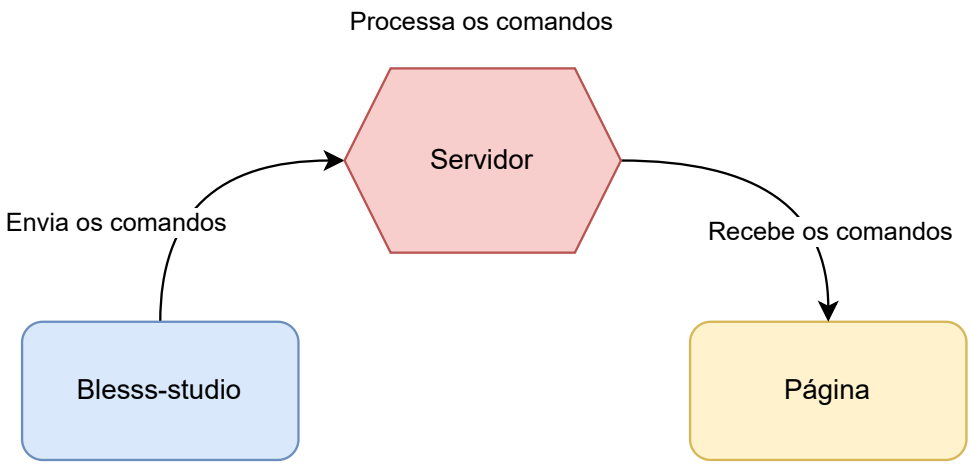
3.3.3 Arquitetura de funcionamento de uma Página

Com base nos apontamentos descritos, é necessário demonstrar toda a arquitetura necessária para que seja possível o funcionamento da estrutura projetada para a representação de uma Página na disposição de Eventos da plataforma Blesss.

Para dar início a este respectivo assunto, é importante fazer uma breve menção ao Blesss-studio, sistema administrador de toda a plataforma em questão. Este sistema possui uma grande importância neste processo, pois é através do mesmo que são inciadas as *lives* das Páginas, assim como também é feito o controle do *chat* de mensagens da plataforma.

Uma vez introduzido o sistema Blesss-studio a Figura 3.7 representa o diagrama arquitetural da comunicação base realizada entre o sistema administrador e a representação de uma Página abordada previamente. Observando o diagrama é possível analisar que o sistema Blesss-studio envia as informações de controle para o servidor, que por sua vez, envia as informações via *socket* para a respectiva Página.

Figura 3.7 – Comunicação entre o sistema Blesss-studio e uma Página

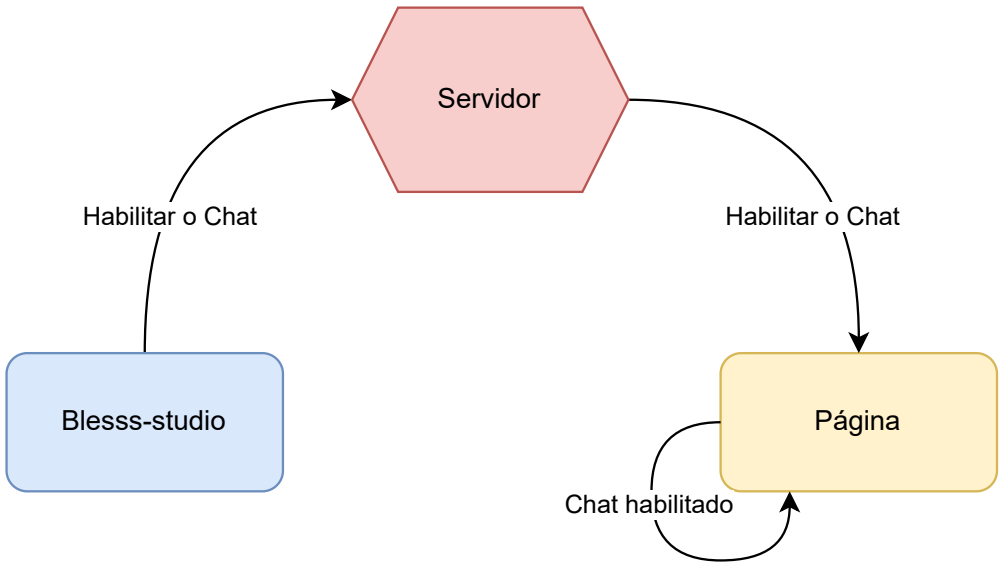


Fonte: do autor

O diagrama faz alusão a intercomunicação geral entre os dois sistemas, e dessa forma, o fluxo de funcionamento de uma palestra é dado de modo que *streaming* da mesma é iniciado com o uso do Blesss-studio, que possui toda uma estrutura desenvolvida para receber o canal de vídeo e enviar para o servidor. Deste modo o servidor por sua vez processa as informações do vídeo e envia os dados para a Página, informando a mesma o atual estado que se encontra o *streaming*, que como apresentado na Figura 3.6 é sequencial e inicia como “Unstarted”.

O *chat* de mensagens presente na Página também segue uma arquitetura semelhante a da Figura 3.7, a diferença básica é que no Blesss-studio também existe um *chat*, que é espelho do existente na respectiva Página, deste modo, os controles ficam centralizados na plataforma administradora, favorecendo o controle e monitoramento do mesmo. É importante ressaltar que tanto os estados da *live*, quanto o *chat*, seguem a arquitetura de comunicação elencada anteriormente (Figura 3.8).

Figura 3.8 – Atualização das informações de uma Página



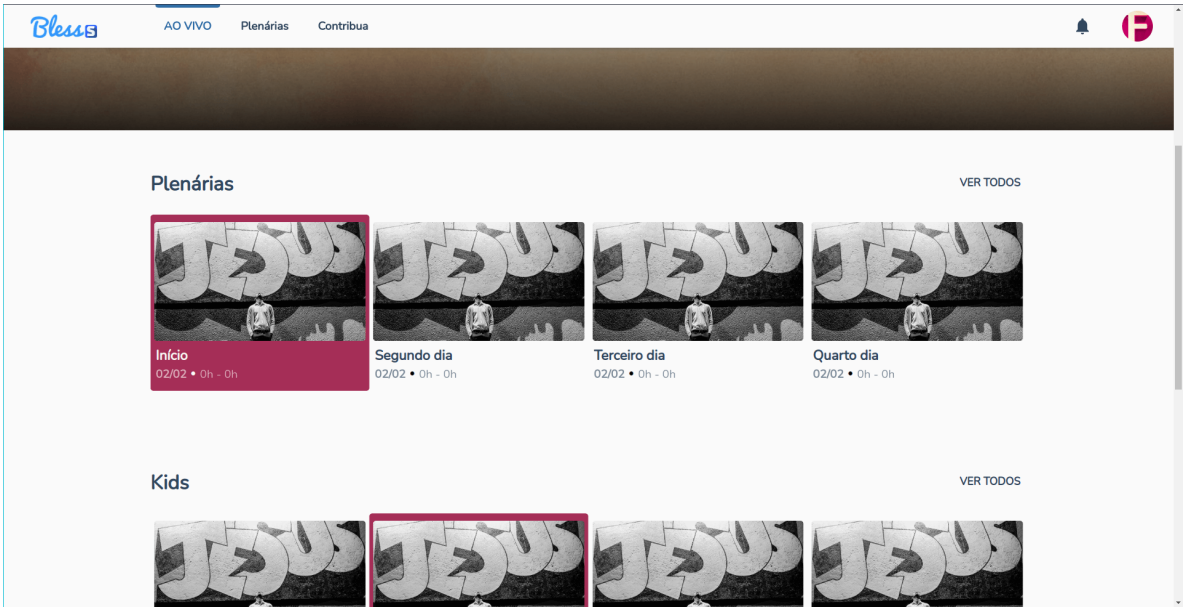
Fonte: do autor

3.4 Implementação

Com base em toda a concepção e planejamento abordado nas seções anteriores, esta seção visa apresentar a etapa de desenvolvimento responsável pelo módulo de Eventos da plataforma Blesss, em especial da representação de Página. Os resultados gerados nas páginas web externas a representação de Página são apresentados, porém como dito anteriormente, este trabalho não apresenta o processo

de desenvolvimento da Playlist e Módulos como um todo. A partir deste levantamento, a Figura 3.9 apresenta o resultado obtido dessas entidades, em que é possível observar a Playlist Consciência Cristã, assim como seus respectivos Módulos (Plenárias e Kids).

Figura 3.9 – Página de Evento, juntamente com Módulos e Páginas



Fonte: do autor

As principais tecnologias utilizadas no desenvolvimento do projeto foram: O Next.js, um *framework* do React JS, para criar todos os componentes e as interfaces do projeto; Material UI, a biblioteca UI de onde todos os componentes bases foram retirados (botões, ícones, entradas de texto); Axios, uma biblioteca para realizar chamadas a API do Back-end em todos os componentes; e Redux, uma biblioteca para controlar o armazenamento dos dados consumidos pelos componentes de uma forma geral. Estas tecnologias são descritas com detalhes no Capítulo 2.

Para dar início a criação da representação de Página foi utilizado o Next.js. O objetivo desta tecnologia é possibilitar o desenvolvimento de interfaces baseadas em *server-side render*, que proporciona uma otimização extra visto que lidar com *streaming* de dados é um assunto que exige mais recursos que páginas web mais simples.

Primeiramente, após definida a base com o uso do Next, foi utilizado o Axios para que uma requisição fosse feita a fim de trazer as informações da Página como um todo, informações essas que são apresentadas na Figura 3.4. Assim que as informações são obtidas, as mesmas são armazenadas em um estado global do Redux para que todos os componentes possam consumi-las.

Para a construção dos componentes da Página, as Subseções seguintes abordam cada um deles, assim como suas formas baseadas no estado da *live*, melhor explicitado na Subseção 3.3.2

3.4.1 Construção do componente Player de vídeo

A construção do componente de vídeo da Página teve como base o uso da biblioteca Clappr [3], que é uma ferramenta de reprodução de vídeo que oferece várias possibilidades de personalização, além de ser portátil para diferentes plataformas.

Inicialmente, foi importado uma instância do Clappr e adicionado um controlador a fim de criar um *player* genérico. A partir disso, o objetivo da mesma era permitir o gerenciamento do *player* de vídeo como um todo pelo *status* da *live*. Este controlador é uma camada de *socket* que utiliza a biblioteca Socket.io, e que também segue o modelo de arquitetura explicado na Subseção 3.3.3 para receber as informações do *status* em questão.

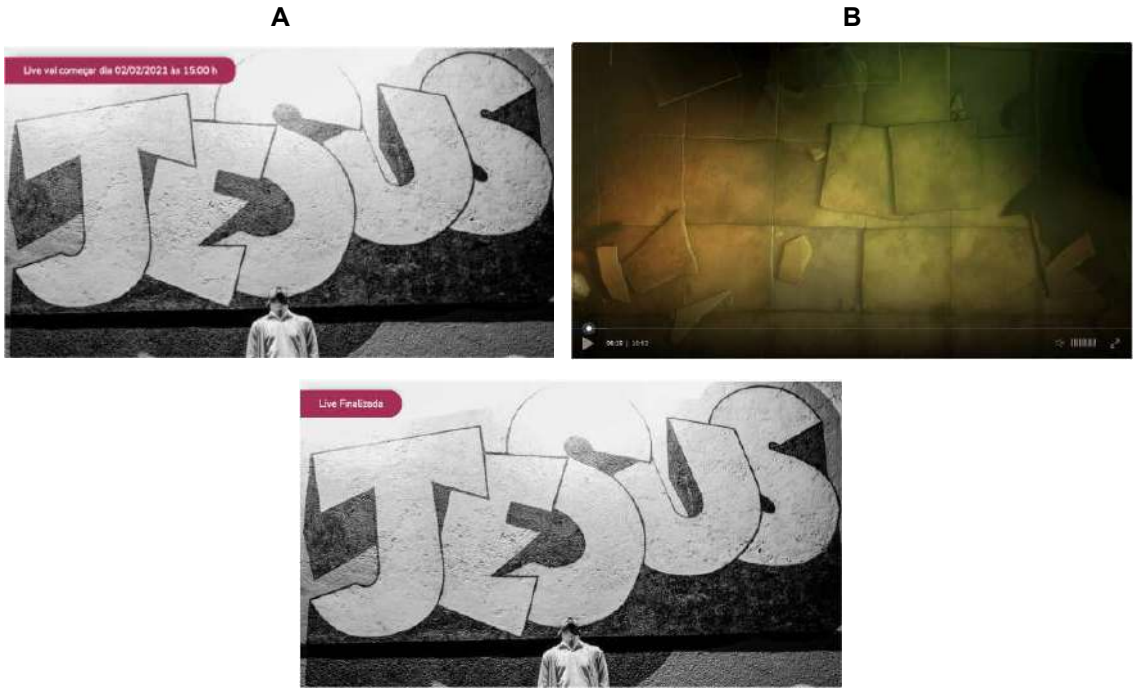
Em seguida, foi criado um elemento que apresenta a data de início da palestra em questão, e também se a mesma já foi finalizada. Este componente é uma *tag* que também muda de acordo com o *status* da *live*.

O componente geral resultante da união dos demais componentes mencionados anteriormente foi o *player* de vídeo. O funcionamento geral do mesmo é detalhado na Seção 3.3.2. Porém, quando o *status* se encontra com o valor “Started”, o *player* recebe o *link* do servidor de *streaming* gerado pela plataforma Blesss-studio via Socket.io, e dessa maneira, a reprodução da palestra em questão é iniciada. Outro ponto importante a ser mencionado, é que o componente de *tag* somente aparece quando o *status* possuir os valores “Unstarted” ou “Finished”, alegando que a palestra ainda vai ocorrer, ou se já foi finalizada.

[3] http://clappr.io/

Os resultados finais do componente *player* de vídeo seguindo os repectivos estados “Unstarted” (**A**), “Started” (**B**) e “Finished” (**C**) podem ser acompanhados mediante a Figura 3.10.

Figura 3.10 – Player de vídeo nos seus respectivos estados



Fonte: do autor

3.4.2 Construção do componente Chat de mensagens

Em relação aos demais componentes presentes na Página, a criação do componente de *chat* foi relativamente mais complexa, uma vez o mesmo necessitava de uma diversidade de validações, sendo elas de entrada, e também, de controles relacionados ao *status* da *live* de uma Página. Assim como no componente *player* de vídeo, as mensagens chegavam através da camada de *socket* da Página, de modo que quando o *status* da *live* estivesse como “Started”, era iniciado o fluxo de recebimento conforme

as mesmas eram enviadas. É valido ressaltar que a mesma camada de *socket* utilizada no *player* de vídeo (controlador) foi inserida no *chat*. Um ponto extra de tratamento que foi considerado, estava na situação de quando um usuário abrisse a Página que já estava com a palestra em andamento, o mesmo deveria receber as mensagens antigas que foram enviadas anteriormente. Logo, foi implementado na camada de *socket* um controle que funcionava de modo que assim que um usuário abrir uma Página com *status* "Started", o mesmo iria receber as mensagens antigas, e deste modo, as mesmas seriam concatenadas com as demais que chegasse a partir daquele momento.

Para dar início a construção da parte visual, um componente de mensagens foi criado. O objetivo deste estava voltado para a representação das mensagens enviadas pelo usuário. Inicialmente foi criado um componente React que recebia alguns parâmetros, sendo eles a identificação do usuário que enviou, o nome do mesmo, o texto da mensagem, a cor do nome do usuário, e por último, um valor *booleano* que informava se a mensagem foi enviada do Blesss-studio. Caso o valor do *booleano* estivesse como verdadeiro, significava que a mensagem foi enviada por um usuário administrador, sendo assim, o corpo da mensagem deveria ser diferente das que foram enviadas por usuários comuns.

O estilo do componente de mensagem foi desenvolvido utilizando CSS, de modo que o nome do usuário e o texto da mensagem ficassem posicionados horizontalmente. Como dito anteriormente, a cor do nome da mensagem era obtida por meio da camada de *socket* e o texto da mensagem recebeu um estilo diferente a fim de padronizar as cores dos textos das mensagens. Um estilo diferente foi desenvolvido no caso de quando a mensagem fosse de um usuário administrador, em que a cor do corpo da mensagem receberia a cor azul, o nome do usuário sempre seria amarelo, e também, foi utilizado um ícone do Material UI para representar o usuário administrador.

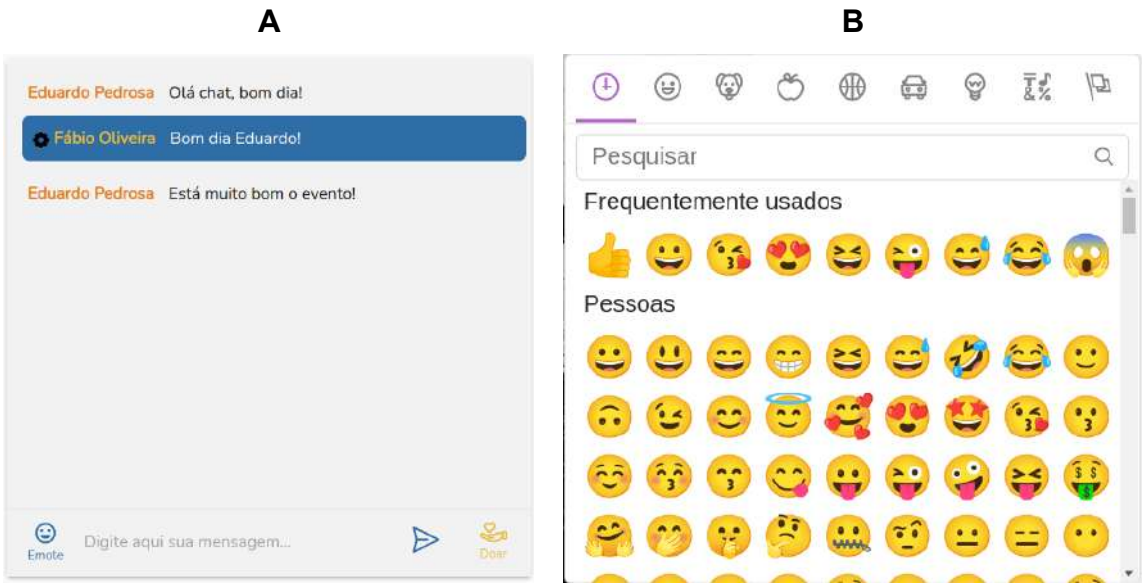
O Próximo componente que foi criado foi o *container* das mensagens, o mesmo era responsável por encapsular todas as mensagens, e tinha a finalidade de funcionar como um *scroll* infinito. Um comportamento único deste componente estava na sua capacidade de dar um auto *scroll* para baixo conforme as mensagens chegavam. Para isso foi criada uma função no controlador que realizava essa ação.

Por fim foi criado o componente de *input* do *chat*, que era responsável pela entrada de texto no mesmo. Esse componente foi criado com a utilização do elemento de entrada de texto do Material

UI, e também seus respectivos ícones, além da utilização de uma biblioteca de *emojis* que foi utilizada para oferecer ao usuário a possibilidade de mandar mensagens mais personalizadas. O componente era formado por quatro partes, um botão de *emojis*, uma entrada de texto, o botão de enviar a mensagem, e por fim, um botão que levava para uma plataforma externa de doação. Os botões de ação foram criados através de ícones do Material, e estilizados com CSS, de modo que todos ele se apresentassem em uma linha horizontal.

Razões importantes a serem consideradas, estão na validação da mensagem do usuário, que poderia conter no máximo 500 caracteres, e também, na capacidade da entrada de texto em informar para o usuário sobre o *status* da *live*, por meio das mensagens“Esperando live começar”quando o valor do *status* fosse “Unstarted”, “Digite uma mensagem...” quando o mesmo era “Started”, e“Live finalizada” quando a palestra estivesse terminado. Uma outra mensagem importante era a “Você foi bloqueada pelo administrador”, quando o usuário administrador bloqueasse um usuário. É válido ressaltar que as mensagem apareciam no *placeholder* do *chat*. Os resultados geral da união dos componentes mencionados anteriormente podem ser melhor compreendidos mediante a Figura 3.11, que representa um *chat* de uma palestra em andamento (**A**), assim como o componente de *emojis* oriundo do clique no botão denominado “Emote” (**B**).

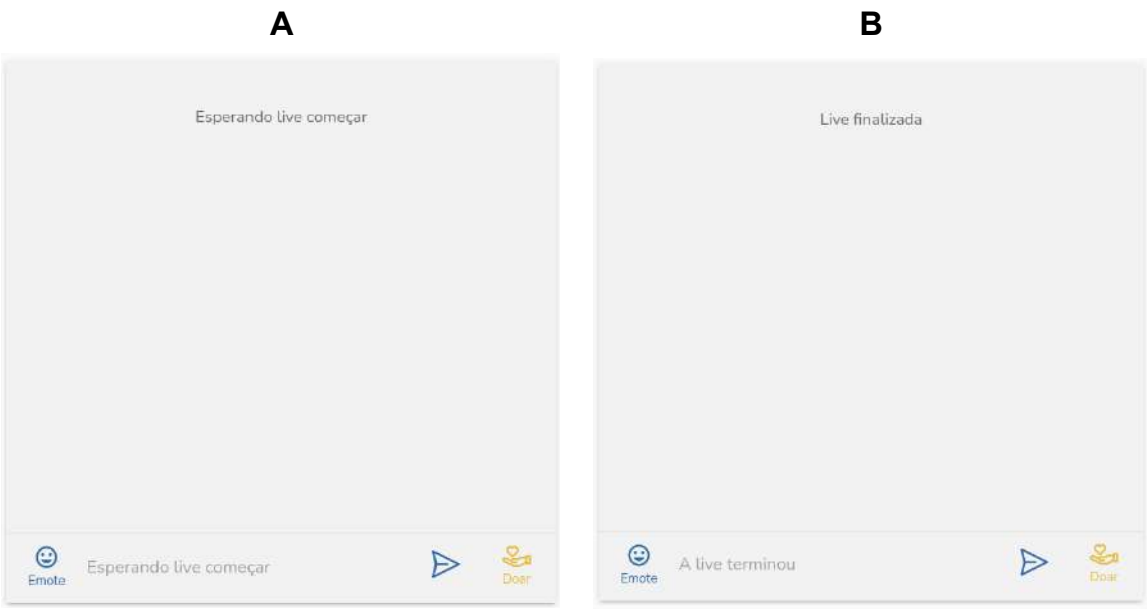
Figura 3.11 – *Chat* de mensagens e *emojis*



Fonte: do autor

De acordo com o *status* da *live*, alguns outros comportamentos do componente de *chat* foram desenvolvidos a fim de sanar os respectivos estados “Unstarted” e “Finished”. Estes comportamentos foram alterações de texto no componente que encapsula as mensagens, a fim de informar o usuário que a live ainda não tinha sido iniciada, ou que a mesma havia sido finalizada. A Figura 3.12 representa estes respectivos comportamentos anteriormente mencionados (“Unstarted” - **A**, “Finished” - **B**).

Figura 3.12 – *Chat* de mensagens nos estados “Unstarted” e “Finished”



Fonte: do autor

3.4.3 Construção do componente Behavior

De acordo com a definição dos componentes de uma Página aborda na Subseção 3.3.1, o componente Behavior possui a finalidade de oferecer ao usuário algumas possíveis interações aos usuários (gostar, não gostar, compartilhar e também relatar possíveis problemas presentes na live). Com isso, o mesmo contém algumas responsabilidades além das interações, sendo elas informar ao usuário o nome da Página, os preletores que estão ministrando as palestra, além do horário de início e fim da transmissão.

Conforme mencionado no início desta Seção, quando uma Página é aberta, uma requisição é realizada (explicitada na Subseção 3.2.1), e nela contém diversas informações da respectiva Página, como nome, preletores, data e hora de início e de fim, entre outras. Essas informações são utilizadas para a construção do componente Behavior como um todo.

Para dar início na construção do componente, um *container* externo foi criado a fim de armazenar toda a estrutura do elemento em questão no seu interior, deste modo, foi adicionado textos para que fosse possível exibir o nome da palestra e também a data de início e fim da mesma. Em seguida, foi criado um outro componente responsável por mapear os preletores, de modo que o *avatar* e o nome aparecesse em linha. Dessa maneira, a lista de preletores era exibida. O *container* continha um comportamento de ampliação e também redução, ambos controlados por um clique em um botão com um ícone de seta para baixo.

Por fim foi criado um componente de interações, que teria quatro botões com os textos "Gostei", "Não gostei", "Compartilhar" e "Relatar", todos os ícones desses respectivos botões foram utilizados da biblioteca Material UI. Os botões "Gostei" e "Não gostei" continham comportamentos semelhantes que quando clicados ambos mudavam de cor para azul, elemento que informava ao usuário que sua interação foi guardada. O resultados desse cliques geravam o disparos de *actions* do Redux que enviavam requisições para o *Back-end* para que as interações do usuário pudessem ser armazenadas no banco de dados. Os outros dois botões quando clicados exibiam modais para o usuário, de modo que um era responsável por exibir informações relacionadas com o compartilhamento da Página (ação já existente na plataforma Blesss) e o outro exibia um modal com possíveis problemas que pudessem estar acontecendo na *live*.

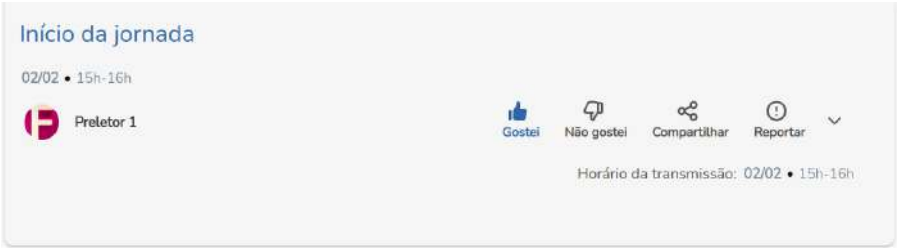
As opções apresentadas no modal de reportar problemas funcionava de maneira parecida com os botões de interações, que quando clicadas, as mesmas disparavam *actions* para do Redux, que por sua vez, enviavam requisições para o *Back-end* informando os possíveis problemas, que por sua vez eram exbidos na plataforma Blesss-studio para os administradores. A listagem de problemas continham opções de relatar problema de título da Página, problemas no vídeo, problemas de aúdio e problemas de conexão.

A estilização dos textos foram foram feitas utilizando CSS, e na estruturação dos componentes foi utilizado Material UI. Para exibir o *avatar* dos preletores foi importando do Material um elemento de exibição de imagens em formato de círculo. Os componentes foram posicionados de modo que o nome e os horários de início e fim ficaram acima da exibição de preletores, e o componente de intera-

ções ficassem a direita dos preletores. Para finalizar, quando o *container* externo estivesse ampliado, era exibido um texto mais detalhado em relação ao horário da transmissão.

Os resultados gerados da construção do componente como um todo podem ser verificados nas Figuras 3.13 e 3.14, em que é demonstrado o componente Behavior e o modal de relatar problemas.

Figura 3.13 – Componente Behavior



Fonte: do autor

Figura 3.14 – Modal de reportar problemas em uma Página

Fonte: do autor

3.4.4 Construção do componente de Exibição de Páginas

Assim como mencionado na Seção 3.3.1, o componente de exibição de Páginas tinha a finalidade de exibir todas às Páginas do respectivo Módulo a fim de facilitar a navegação entre o conteúdo do mesmo. Inicialmente para armazenar o conteúdo foi utilizado um componente de *carrousel* criado

em React previamente utilizado em demais localizações da plataforma como um todo. No geral o componente de *carrousel* funcionava de modo que o mesmo recebia uma lista de dados, e logo em seguida, exibia-os horizontalmente semelhante a uma galeria de fotos.

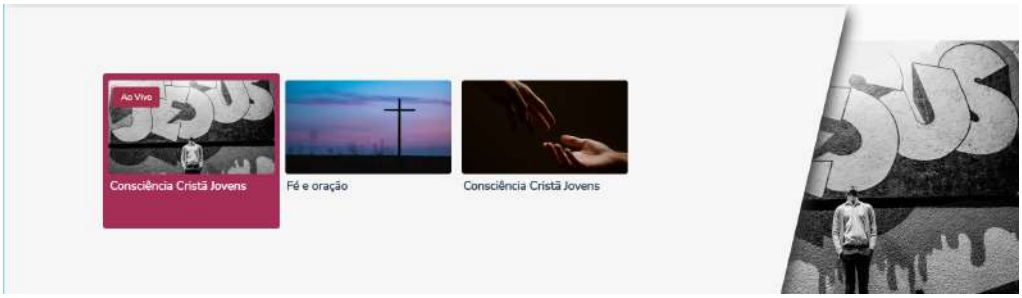
Por intermédio da requisição inicial realizada quando a respectiva Página era aberta (informada no início desta Seção), os dados das Páginas do Módulo eram obtidos e continham informações de *thumbnail*, nome e o *status*, atual estado que a Página se encontrava. Assim como citado, essas informações eram recuperadas através do estado global do Redux, de maneira que essa recuperação acontecia por meio de um disparo de uma *action* que retornava os dados do estado global.

A partir do componente de *carrousel* previamente abordado, foi criado uma extensão do mesmo com adição de uma função para destacar com a cor do Módulo qual era a Página atual que estava sendo exibida, e também, um sistema de *tag* semelhante a do componente de vídeo, que tinha a finalidade de informar o estado atual das Páginas presentes no respectivo *carrousel* de Páginas. A *tag* existente continha o texto “Ao vivo” caso a *live* da Página estivesse *online*, aspecto que informava o usuário caso aquela palestra estivesse sendo ministrada.

Seguindo a construção, foi criado um componente que importava o *carrousel* juntamente com as novas funcionalidades a fim de armazenar estilos necessários para o resultado final do mesmo. Estes estilos foram desenvolvidos com o uso da ferramenta de CSS da biblioteca Material UI, de maneira que foi utilizado o esquema de *grid* para posicionar o *carrousel* no centro, e também, foi utilizado estilos em CSS para deixar a listagem de Páginas em um formato de um trapézio, a fim de seguir o *layout* apresentado pela equipe de *design*. Por fim, a *thumbnail* da respectiva Página exibida naquele momento foi adicionada no lado direito do componente para finalizar a construção do componente como um todo.

O resultado gerado de todas as etapas mencionadas anteriormente pode ser melhor observado na Figura 3.15, que mostra o componente de exibição de Páginas em sua versão final.

Figura 3.15 – Listagem das Páginas de um Módulo

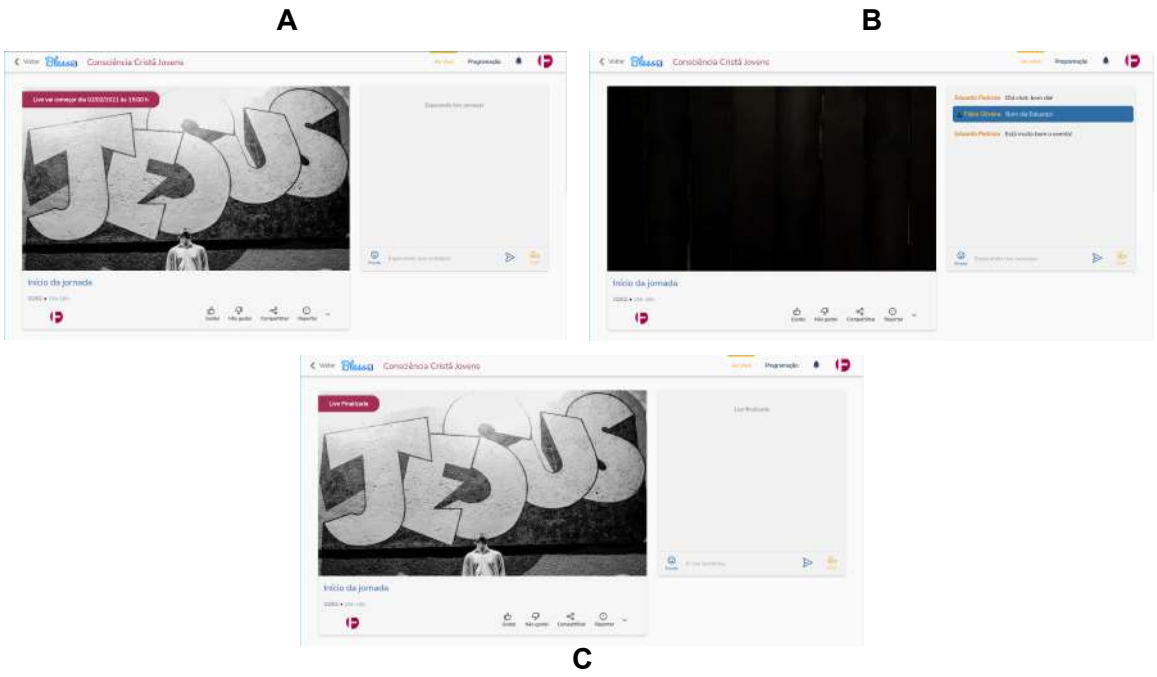


Fonte: do autor

3.5 Resultados gerais

Após a disposição dos componentes, a Página foi montada através da importação de todos em um arquivo, cujo objetivo era estruturar o funcionamento da mesma como um todo. Desta forma, a combinação dos componentes anteriormente elencados trabalhava de acordo com o *status* da Página, e assim, apresentava-se de diferentes maneiras. Com isso, a Figura 3.16, apresentam a Página de acordo com os respectivos *status* “Unstarted” (**A**), “Started” (**B**) e “Finished” (**C**).

Figura 3.16 – Resultado da composição dos componentes



Fonte: do autor

3.6 Considerações finais

Com base no conteúdo discorrido anteriormente, este capítulo apresentou todos os passos necessários para que fosse possível o desenvolvimento da Página presente na estrutura de eventos da plataforma Blesss. Após construída e colocada no ar, a estrutura de eventos concluiu o objetivo de realizar a cobertura da Consciência Cristã em sua forma remota, porém, alguns problemas relacionados a infraestrutura tornou ainda mais desafiador a realização do evento *online*.

Através dos problemas de infraestrutura mencionados, a equipe de desenvolvimento teve a missão de manter todo o ecossistema da plataforma Blesss funcionando para que fosse possível cobrir a Consciência Cristã, motivo que colaborou para o conhecimento e experiência de todos os membros presentes.

Durante o evento a recepção de toda a estrutura foi bem recebida pelos usuários, condição que levou a um pico de mais de 1000 clientes simultâneos fazendo o uso da estrutura durante a exibição do evento Consciência Cristã 2021, dessa forma, todo o objetivo inicial planejado e implementado pode ser concluído, e a estrutura foi mantida na plataforma para cobrir as próximas edições do evento.

Baseado nas implementações discorridas neste capítulo, algumas melhorias podem ser consideradas para trabalhos futuros. A adoção de melhorias na construção do *player* de vídeo é um fator que melhorará muito o desempenho do evento no geral, uma vez que a quantidade de conteúdo de *streaming* carregado no mesmo pode ser diminuída, evitando com que requisições desnecessárias sejam realizadas no servidor, ação que reduz a sobrecarga geral da aplicação.

4 CONCLUSÃO

Este documento descreveu as atividades desenvolvidas pelo autor no contexto do estágio supervisionado realizado na empresa Zeester, no perído de 07/2020 à 08/2021. O estágio consistiu no desenvolvimento do módulo de eventos da plataforma Bless, principal software mantido pela empresa, cujo objetivo foi apoiar o evento da Consciência Cristã 2021.

O estágio proporcionou a aquisição de uma diversidade de conhecimento, mediante a experiência de acompanhar de perto como funciona o planejamento, o projeto e a criação de sistemas Web que visam solucionar problemas reais existentes no mercado de trabalho.

De acordo com todas as experiências adquiridas durante a realização do estágio, foi possível notar o quanto é importante, para um aluno de graduação, o contato com problemas reais existentes, uma vez que é através dos mesmos que conceitos adquiridos na graduação podem ser aplicados e vistos em funcionamento na prática. Com base nesta condição, a formação profissional do estagiário teve uma grande acréscimo, fator que colaborou e resultou na contratação do mesmo por uma grande empresa de tecnologia situada no Rio de Janeiro. Olhando para questões pessoais, o estagiário pode realizar um grande sonho da sua vida, que era ingressar no mercado de trabalho com o conhecimento necessário, e com isso, crescer profissionalmente através do seu esforço.

Contudo, também é válido ressaltar a importância da formação acadêmica, que sem dúvidas faz uma grande diferença quando o assunto é lidar com problemas reais existentes. Uma diversidade de disciplinas foram importantes, porém, introdução aos algoritmos, programação Web e modelagem e implementação de software foram disciplinas que ampliaram a todos os conhecimentos necessários para que fosse possível realizar o estágio. Este ponto foi nítido durante a realização do estágio, uma vez que todos os conceitos vistos nas disciplinas da graduação puderam ser colocados em prática de uma maneira mais direta no estágio.

Por fim, é interessante ressaltar que por mais que muitas experiencias puderam ser vividas pelo estagiário na graduação, o mesmo sentiu um pouco de falta de não ter participado de grupos de estudos científicos, e também, estudos dirigidos para o mercado de trabalho, princípio que colaboraria bem mais para sua formação acadêmica e para sua vida no mercado de trabalho como um todo. A graduação foi uma das melhores experiências vividas pelo estagiário, que pode ter vários contato com

diversos professores e profissionais atuantes na área, motivo que levou o mesmo a se dedicar em aprender todos os conceitos vistos em sala de aula.

REFERÊNCIAS

AGGARWAL, S. Modern web-development using reactjs. **International Journal of Recent Research Aspects**, v. 5, n. 1, p. 133–137, 2018.

AXIOS. **Getting Started**. 2022. Disponível em: <https://axios-http.com/docs/intro>. Acesso em: 8 jun. 2022.

BANGARE, S. et al. Using node. js to build high speed and scalable backend database server. In: **Proc. NCPCI. Conf**. [S.l.: s.n.], 2016. p. 19.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. **Hypertext transfer protocol–HTTP/1.0**. [S.l.], 1996.

Blesss. **Blesss**. Zeester, 2022. Disponível em: <https://blesss.org/meublesss>. Acesso em: 07 jul. 2022.

BODUCH, A. **React Material-UI Cookbook: Build captivating user experiences using React and Material-UI**. [S.l.]: Packt Publishing Ltd, 2019.

CANIUSE. **JavaScript | Can I use... Support tables for HTML5, CSS3, etc**. 2022. Disponível em: <https://caniuse.com/?search=JavaScript>. Acesso em: 19 maio. 2022.

CHACON, S.; STRAUB, B. **Pro git**. [S.l.]: Springer Nature, 2014.

CIOTTI, M. et al. The covid-19 pandemic. **Critical reviews in clinical laboratory sciences**, Taylor & Francis, v. 57, n. 6, p. 365–388, 2020.

COULOURIS, G. et al. **Sistemas Distribuídos-: Conceitos e Projeto**. [S.l.]: Bookman Editora, 2013.

DRESHER, T.; ZUKER, A.; FRIEDMAN, S. **Hands-On Full-Stack Web Development with ASP. NET Core: Learn end-to-end web development with leading frontend frameworks, such as Angular, React, and Vue**. [S.l.]: Packt Publishing Ltd, 2018.

FLANAGAN, D. **JavaScript: o guia definitivo**. [S.l.]: Bookman Editora, 2004.

HAZZAN, O.; DUBINSKY, Y. **Agile software engineering**. [S.l.]: Springer Science & Business Media, 2009.

JING, J.; HELAL, A. S.; ELMAGARMID, A. Client-server computing in mobile environments. **ACM computing surveys (CSUR)**, ACM New York, NY, USA, v. 31, n. 2, p. 117–157, 1999.

KONSHIN, K. **Next. js Quick Start Guide: Server-side rendering done right**. [S.l.]: Packt Publishing Ltd, 2018.

KUMAR, G.; BHATIA, P. K. Impact of agile methodology on software development process. **International Journal of Computer Technology and Electronics Engineering (IJCTEE)**, v. 2, n. 4, p. 46–50, 2012.

LECHETA, R. R. **Web Services RESTful: Aprenda a criar web services RESTful em Java na nuvem do Google**. [S.l.]: Novatec Editora, 2015.

MURPHY, L. et al. Api designers in the field: Design practices and challenges for creating usable apis. In: IEEE. **2018 ieee symposium on visual languages and human-centric computing (vl/hcc)**. [S.l.], 2018. p. 249–258.

NODE.JS. **Node.js**. 2022. Disponível em: <https://nodejs.org/pt-br/>. Acesso em: 26 maio. 2022.

OLUWATOSIN, H. S. Client-server model. **IOSRJ Comput. Eng**, v. 16, n. 1, p. 2278–8727, 2014.

PEREIRA, C. R. **Aplicações web real-time com Node. js**. [S.l.]: Editora Casa do Código, 2014.

PORTELA, C. F.; QUEIRÓS, R. **Introdução ao desenvolvimento moderno para a Web**. [S.l.]: FCA Editora, 2018.

Q-SUCCESS. **Usage statistics of JavaScript as client-side programming language on websites**. 2022. Disponível em: <https://w3techs.com/technologies/details/cp-javascript/>. Acesso em: 19 maio. 2022.

RAI, R. **Socket. IO Real-time Web Application Development**. [S.l.]: Packt Publishing Ltd, 2013.

REACT. **React**. Facebook, 2022. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 3 jun. 2022.

REDUX. **Redux - A predictable state container for JavaScript apps**. Redux, 2022. Disponível em: <https://redux.js.org/>. Acesso em: 4 jun. 2022.

RUBIN, K. **Essential Scrum: A Practical Guide to the Most Popular Agile Process**. Pearson Education, 2012. (Addison-Wesley Signature Series (Cohn)). ISBN 9780321700377. Disponível em: <https://books.google.com.br/books?id=3vGEcOfCkdwC>.

Scrum Guide. **Scrum Guide**. 2022. Disponível em: <https://scrumguides.org/scrum-guide.html>. Acesso em: 14 jun. 2022.

SOCKET.IO. **Introducion**. 2022. Disponível em: <https://socket.io/docs/v4/>. Acesso em: 8 jun. 2022.

SOMASUNDARAM, R. **Git: Version Control for Everyone Beginner's Guide**. [S.l.]: Packt Pub., 2013.

UI, M. 2022. Disponível em: <https://mui.com/pt/>. Acesso em: 04 jun. 2022.

V8. **V8 JavaScript engine**. 2022. Disponível em: <https://v8.dev/>. Acesso em: 26 maio. 2022.

VERCEL. **Next.js**. Vercel, 2022. Disponível em: <https://nextjs.org/learn/basics/create-nextjs-app>. Acesso em: 8 jun. 2022.

VINACC. **Principal - Landing Page VINACC**. VINACC, 2022. Disponível em: <https://vinacc.org.br/>. Acesso em: 09 jul. 2022.

VSKILLS. **Node.JS Architecture - Tutorial**. VSKILLS, 2022. Disponível em: <https://www.vskills.in/certification/tutorial/node-js-architecture-2/>. Acesso em: 7 jun. 2022.