GUILHERME HENRIQUE PENA SOUSA

DESENVOLVIMENTO E MANUTENÇÃO DO SISTEMA BLESSS NA EMPRESA ZEESTER:

PLATAFORMA STREAMING BLESSS

LAVRAS – MG

2022

GUILHERME HENRIQUE PENA SOUSA

DESENVOLVIMENTO E MANUTENÇÃO DO SISTEMA BLESSS NA EMPRESA ZEESTER:

PLATAFORMA STREAMING BLESSS

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

Prof. Dr. Raphael Winckler De Bettio

Orientador

LAVRAS – MG

2022

GUILHERME HENRIQUE PENA SOUSA

DESENVOLVIMENTO E MANUTENÇÃO DO SISTEMA BLESSS NA EMPRESA **ZEESTER**: PLATAFORMA STREAMING BLESSS

DEVELOPMENT AND MAINTENANCE OF BLESSS SYSTEM AT ZEESTER: BLESSS STREAMING PLATFORM

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

APROVADA em 14 de Agosto de 2022.

Prof. Dr. Eric Fernandes De Mello Araujo
Prof. DSc. Renata Teles Moreira

gov.br
Documento assinado digitalmente
RAPHAEL WINCKLER DE BETTIO
Data: 23/09/2022 09:47:56-0300
Verifique em https://verificador.iti.br

Prof. Dr. Raphael Winckler De Bettio
Orientador

LAVRAS – MG
2022

RESUMO

Este documento tem como finalidade relatar as atividades realizadas durante determinado período de estágio exercido na empresa Zeester, as quais consistiam no desenvolvimento de um sistema responsável por servir a uma plataforma de *streaming*. Fundada em 2015 a referida organização tem como foco principal a comunidade cristã e, visando alcançar o maior número de fiéis, criou-se um *software*, o Blesss, responsável por transmitir e compartilhar os ensinamentos religiosos. As principais atividades executadas consistiam no desenvolvimento *back-end* de novas funcionalidades, bem como realizar a manutenção do principal *software* outrora mencionado. Tais tarefas englobavam a melhoria na estrutura, organização do sistema e, por fim, implementação de atividades necessárias a permitir o funcionamento estável da plataforma adaptando-se às necessidades dos usuários.

Palavras-chave: *Back-end.* Desenvolvimento. Sistema

ABSTRACT

The purpose of this document is to report the activities performed during a certain internship period at Zeester, which consisted in the development of a system responsible for serving a streaming platform. Founded in 2015, this organization has as its main focus the Christian community and, in order to reach the largest number of believers, a software, Blesss, was created to transmit and share religious teachings. The main activities performed consisted in the back-end development of new features, as well as the maintenance of the main software mentioned above. Such tasks included improving the structure, organizing the system and, finally, implementing the necessary activities to allow the stable operation of the platform by adapting it to the users' needs.

Keywords: Back-end. Development. System.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A empresa Zeester foi criada em 2015 com o intuito de oferecer um serviço de criação e Desenvolvimento de Negócios e, atualmente, é dividida em cinco equipes que contribuem para o desenvolvimento e manutenção dos projetos realizados, sendo elas: liderança, *design*, desenvolvimento, *marketing* e comunicação. O autor foi alocado na equipe de desenvolvimento como *back-end*, e todas as atividades que serão apresentadas neste documento foram desenvolvidas dentro do escopo desta área.

Embora a equipe de desenvolvimento seja responsável por uma área específica dentro da empresa, ela está conectada diretamente com as equipes de *design*, liderança e comunicação. Através da equipe de liderança, as demandas são criadas e as atividades realizadas pela equipe de desenvolvimento são validadas. A equipe de *design*, por sua vez, disponibiliza fluxos e modelos para a equipe de desenvolvimento ajudando no direcionamento e execução das atividades. A equipe de comunicação é responsável por enviar os erros relatados pelos usuários para a equipe de desenvolvimento, e, por fim, assim que esses são corrigidos, a equipe de desenvolvimento entra em contato com a equipe de comunicação, que avisa aos usuários que o mesmo foi reparado.

Nesse diapasão, o foco principal é oferecer soluções tecnológicas na área de ensino religioso, auxiliando igrejas e comunidades a terem um maior alcance na propagação de seus valores teológicos e incentivar mais pessoas a conhecerem a vida cristã e suas práticas.

Assim sendo, foi criado um novo projeto, denominado Blesss. Trata-se de um sistema responsável por disponibilizar o conteúdo cristão através de vídeos, lives, livros, artigos, questionários e apostilas com o objetivo de engajar mais pessoas a iniciarem o estudo religioso e oferecer um ambiente propício para tal. O intuito deste trabalho é dissertar a respeito das atividades realizadas durante o período de estágio de um ano na referida instituição, bem como, apresentar sua organização de equipes e o desenvolvimento de seu produto.

As tarefas foram desenvolvidas conforme os requisitos e necessidades designadas para a plataforma Blesss. Entre elas estão incluídas: desenvolvimento de novas funcionalidades, desenvolvimento de microsserviços, otimização da estrutura de banco de dados, refatoração de código e correção de problemas encontrados.

2 FERRAMENTAL

Este capítulo tem como objetivo apresentar as tecnologias utilizadas para o desenvolvimento da plataforma Blesss[1], juntamente com as ferramentas utilizadas para ajudar na organização da equipe e desenvolvimento das tarefas.

2.1 JavaScript

JavaScript é uma linguagem de script leve, orientada a objetos e multiplataforma, frequentemente usada em páginas da web (W3TECHS, 2022). Trata-se de sistemas que funcionam em navegadores, como plataformas de gestão e qualquer outra ferramenta com acesso por meio da web (SOUZA, 2019).

(L, 2019) O uso da linguagem JavaScript oferece diversas vantagens, como :

- Alta compatibilidade com plataformas, sistemas e navegadores web;
- É mais leve e rápida que outras linguagens de programação;
- Faz com que as páginas na internet sejam mais dinâmicas e interativas, características essenciais do UX;
- Os navegadores interpretam a linguagem por conta própria, tirando a necessidade de usar um compilador;
- Alta compatibilidade com plataformas, sistemas e navegadores web;
- Erros de programação são mais fáceis de encontrar e de corrigir;
- Entre as mais populares, é a linguagem de programação mais fácil de aprender;
- Executa comportamentos específicos em uma página, como cliques e efeitos personalizados;

Assim como toda linguagem, JavaScript também contém algumas desvantagens que devem ser levadas em conta por um desenvolvedor. As desvantagens mais significativas são (L, 2019):

- É vulnerável à brechas de segurança nos sistemas, navegadores e páginas envolvidas;

[1] <https://blesss.org/>

- Pode ser usada para executar programas maliciosos sem que o usuário saiba;
- Pode ser renderizada de maneiras diferentes pelos dispositivos compatíveis, causando problemas de desempenho;
- Nem sempre compatível com todos os navegadores e sistemas existentes;

2.2 API REST

APIs (*Application Programming Interfaces*) são mecanismos que permitem que dois componentes de *software* se comuniquem usando um conjunto de definições e protocolos. Uma API geralmente funciona operando dois lados, o lado do servidor e o lado do cliente (AMAZON, 2022b). O cliente pode mandar diversas requisições para uma API e o servidor é responsável por enviar uma resposta para cada requisição realizada.

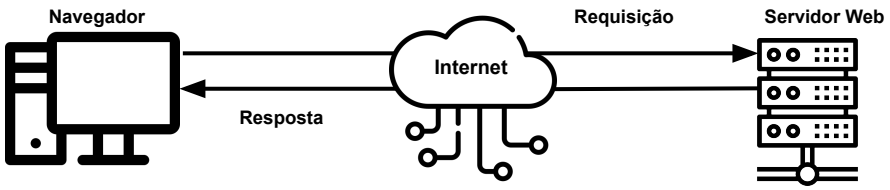
API REST, também chamada de API RESTful, é uma interface de programação de aplicações (API ou API web) que está em conformidade com as restrições do estilo de arquitetura REST, permitindo a interação com serviços web RESTful (HAT, 2020) . O termo REST, é uma sigla que significa transferência de estado representacional.

A principal característica da API REST é a ausência de estado. A ausência de estado significa que os servidores não salvam dados do cliente entre as solicitações (AMAZON, 2022b).

Usar uma API REST pode oferecer várias vantagens para seu projeto, por exemplo : integração, uma vez que ao estar integrando uma API, o desenvolvedor pode aproveitar diversas funcionalidades já presentes no código, não tendo que começar toda vez do zero; Compatibilidade e expansão, já que uma API pode ser executada tanto em um ambiente web quanto em um aplicativo, seja para computador ou para celular.

Na Figura 2.1 é mostrada a representação do fluxo entre o servidor e o cliente em uma API REST.

Figura 2.1 – Fluxo de uma API REST



Fonte: Do próprio autor

2.3 Node.js

Node.js é uma tecnologia usada para executar código JavaScript fora do navegador. Com ela é possível construir aplicações *web* em geral, desde *web sites* até APIs e microsserviços. Isso é possível devido a união do ambiente de execução de JavaScript fornecido pelo próprio Node e o motor de interpretação e execução de JavaScript presente no Google Chrome, chamado de V8 (DEVMEDIA, 2022).

Um dos fatores que fez com que as empresas optassem pelo uso do Node.js no desenvolvimento de seus softwares foram as vantagens que a tecnologia proporcionava, por exemplo: maior leveza no ambiente de desenvolvimento, uma vez que a tecnologia possui um formato *single thread*, o que faz com que ela não utilize tantos recursos computacionais; Outra vantagem é o *Node Package Manager*, que nada mais é do que um repositório com várias bibliotecas prontas para serem utilizadas pelos desenvolvedores, possibilitando uma reutilização de códigos. A flexibilidade oferecida é um dos principais motivos pelo qual as empresas optam por utilizar Node.js, uma vez que ele permite que projetos sejam executados com total flexibilidade entre diferentes sistemas operacionais (LENON, 2018).

2.4 Express

O Express é uma estrutura de aplicativo da Web Node.js mínima e flexível que fornece um conjunto robusto de recursos para aplicativos da *web*. (EXPRESS, 2017)

O uso do Express em uma API pode oferecer algumas vantagens que facilitam o seu desenvolvimento, tais como: possibilitar o tratamento de exceções dentro da própria aplicação, ajudar a gerenciar diversas requisições HTTP e oferecer um sistema completo de rotas.

Na Figura 2.2 é apresentado um exemplo do uso da biblioteca Express.

Figura 2.2 – Iniciando um servidor utilizando express

Fonte: Do próprio autor

Analisando a Figura 2.2, é possível observar na primeira linha que a biblioteca Express está sendo importada no projeto. Logo em seguida, na linha 3, é escolhido uma porta para iniciar o servidor. Na linha 6 é possível observar um simples exemplo de uma rota e por fim na linha 10, o servidor é iniciado na porta selecionada anteriormente.

Na empresa Zeester, a biblioteca Express é utilizada para criar servidores, que possibilitam a comunicação entre os projetos *back-end* e *front-end*, assim como para comunicação entre microsserviços e o projeto principal.

2.5 Scrum

O *Scrum* é um *framework* de trabalho que se baseia nas metodologias ágeis para fazer a gestão de projetos. Ele estabelece uma estrutura interativa e incremental para o desenvolvimento de um produto – seja ele digital ou não (EDUCAÇÃO, 2022).

O principal motivo para utilizar a metodologia *Scrum* no desenvolvimento de um projeto, seja ele de *software* ou não, é facilitar e estimular as equipes envolvidas a trabalharem juntas e de forma organizada, através de reuniões e registro que podem ser feitos durante o desenvolvimento de algum projeto. A cada reunião os participantes tem a chance de relatar problemas encontrados, impedimentos, progressão de uma tarefa e vários outros fatores relevantes que se manifestaram durante a realização das tarefas propostas.

O *Scrum* é estruturado em interações constantes entre os membros das equipes, entregas contínuas com *feedbacks* recorrentes, que proporcionam aprendizado, e, com isso, um desenvolvimento incremental e dinâmico (EDUCAÇÃO, 2022).

As entregas são apresentadas em uma lista, chamada de *Backlog* do produto. Esta lista contém os requisitos necessários para a construção de um produto de alto valor. Ou seja, trata-se de uma lista de entregas do projeto organizada em ordem de prioridade (ESPINHA, 2022).

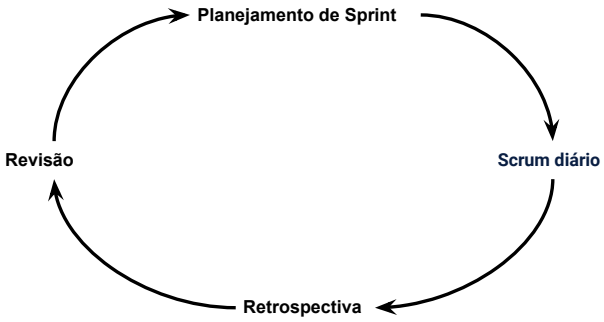
O objetivo da aplicação contínua do *Scrum* é fazer com que os participantes contribuam de uma forma geral com os conhecimentos e experiência que adquiriram durante o desenvolvimento de uma tarefa, contribuindo para o conhecimento geral da equipe, fazendo com que todos os integrantes tenham a chance de conhecer cada vez mais sobre o projeto durante a realização da *Sprint*, adquirindo conhecimento e experiências em todas as partes do projeto.

Assim como é citado em (DRUMOND, 2022b), a estrutura da metologia *Scrum* é dividida em quatro partes que são aplicadas durante o desenvolvimento de uma *Sprint*(Planejamento Da *Sprint*, *Scrum* diário, Retrospectiva e Revisão).

O *Scrum Master* é responsável por garantir que o *Scrum* seja aplicado de maneira correta e impedir interferências externas. Papel de grande destaque porque ele também fica encarregado de mostrar ao time a importância de seguir o método ágil (CONSULTORIA, 2019).

Na Figura 2.3 é representando o fluxo de uma *Sprint* e como ela é dividida.

Figura 2.3 – Fluxo de uma Sprint



Fonte: Do próprio autor

Segundo (CONSULTORIA, 2019), as etapas de uma *Sprint* são definidas da seguinte forma:

- **Planejamento Da *Sprint*:** Nesta etapa são definidas as atividades que serão realizadas pela equipe.
- ***Scrum* diário:** No início de cada dia é feito uma reunião curta, onde é passado como estão sendo executadas as tarefas. O objetivo principal é trazer alinhamento e fazer adaptações caso necessário.
- **Retrospectiva:** É uma reunião onde o time verifica as ações tomadas durante o processo; se foram corretas e o que pode ser melhorado para os próximos ciclos.
- **Revisão:** É uma reunião que acontece após o encerramento de um ciclo de uma *Sprint*. Nesta reunião é validado com o *product owner* se o que foi entregue, corresponde com suas necessidades.

A empresa Zeester utiliza as metodologias oferecidas pelo *framework* na equipe desenvolvimento, com o objetivo de melhorar o gerenciamento das atividades. Um estagiário na equipe de desenvolvimento, utiliza tais metodologias para ajudar no controle de tarefas atribuídas a ele, assim como para mostrar para a equipe a progressão das atividades realizadas.

2.6 Docker

O Docker é uma plataforma de *software* que permite a criação, o teste e a implantação de aplicações rapidamente. O Docker cria pacotes de *software* em unidades padronizadas chamadas de contêineres que têm tudo o que o *software* precisa para ser executado, inclusive bibliotecas, ferramentas de sistema, código e *runtime*. Ao usar o Docker, é possível implantar e escalar rapidamente aplicações em qualquer ambiente e ter a certeza de que o seu código será executado (AMAZON, 2022a)

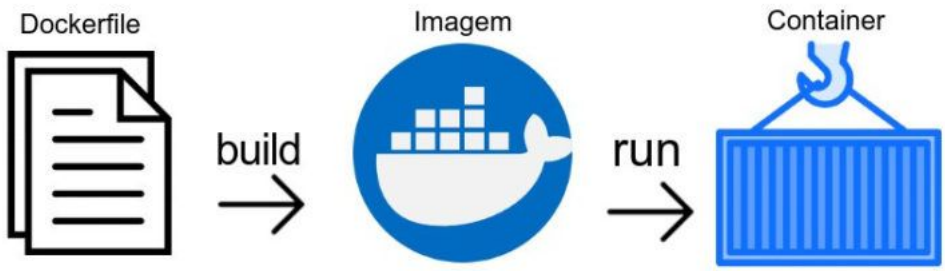
O uso desta plataforma oferece vários benefícios para o desenvolvimento e gerenciamento de projetos de *software*, tais como: maior facilidade para manutenção de código, facilita na criação e escalação de microsserviços, mais agilidade na disponibilização de *software*, maior disponibilidade do sistema e economia de recursos.

Para executar um projeto através do Docker, é necessário seguir o fluxo apresentado na (Figura 2.4) , onde será criado um arquivo chamado de *Dockerfile*, utilizado para configurar e personalizar uma imagem. Em seguida a imagem é construída e por fim é criado um contêiner que poderá ser executado em um sistema.

O *Dockerfile* é um meio que utilizado para criar as próprias imagens. Em outras palavras, ele serve como a receita para construir um contêiner, permitindo definir um ambiente personalizado e próprio para um projeto pessoal ou empresarial (ARTINE, 2019).

A imagem é um arquivo e o contêiner é um processo. Da perspectiva de um sistema operacional, um contêiner é um processo com restrições. No entanto, ao invés de executar um único arquivo binário, um contêiner executa uma imagem. Uma imagem é um pacote de sistema de arquivos que contém todas as dependências necessárias para executar um processo: arquivos de biblioteca, arquivos no sistema de arquivos, pacotes instalados, recursos disponíveis, processos em execução e módulos do *kernel* (TECNISYS, 2019).

Figura 2.4 – Etapas de criação de um container a partir do Dockerfile



Fonte: (ARTINE, 2019)

Na empresa Zeester, a plataforma Docker é utilizada em todas as APIS, com o objetivo de facilitar configuração dos ambientes para os desenvolvedores, oferecendo um acesso prático para qualquer projeto da empresa.

2.7 Visual Studio Code

O Visual Studio Code é um editor de código-fonte leve, mas poderoso, executado na área de trabalho e disponível para Windows, macOS e Linux. Ele vem com o *built-in* para JavaScript, TypeScript e Node.js e possui um rico ecossistema de extensões para outros idiomas e tempos de execução (como C++, C#, Java, Python, PHP, Go, .NET) (MICROSOFT, 2022).

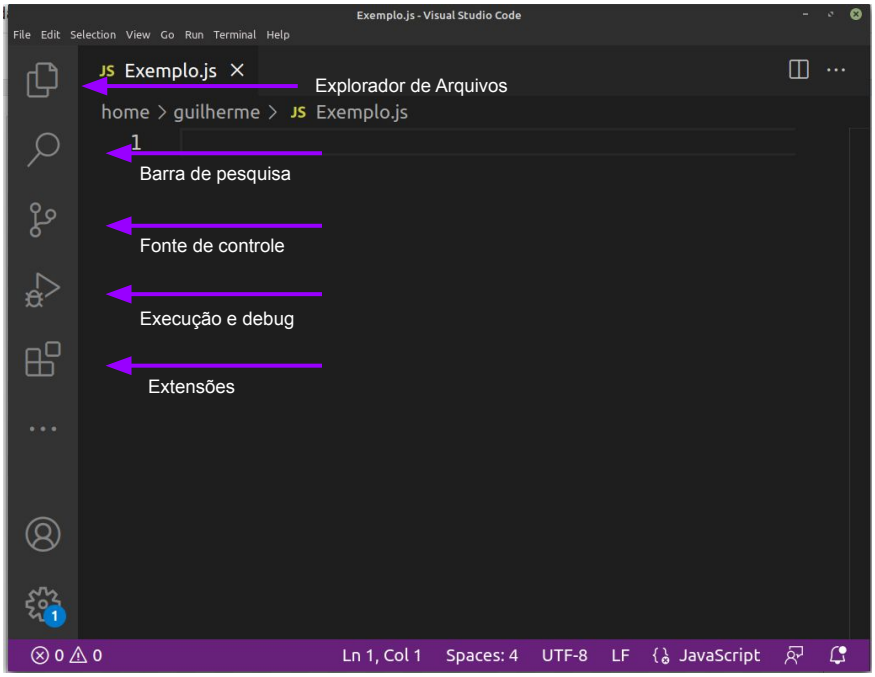
Segundo (HANASHIRO, 2021), existem vários motivos para utilizar o Visual Studio Code como editor de código, tais como:

- Ferramenta simples;
- Código aberto;
- Arquitetura bem planejada;
- Possibilidade de customização;
- Várias funcionalidades e atalhos;
- Rica loja de extensões;

- Facilidade em criar e publicar extensões (escritas em JavaScript ou TypeScript);

Na Figura 2.5 é apresentada a página inicial do Visual Studio Code, com o objetivo de mostrar algumas ferramentas disponíveis na plataforma.

Figura 2.5 – Página incial Visual Studio Code



Fonte: Do próprio autor

Na imagem 2.5 são apresentadas algumas ferramentas disponibilizadas pelo Visual Studio Code, que são:

- **Explorador de Arquivos:** Apresenta todo o caminho de arquivos e pastas de um projeto.
- **Barra de pesquisa:** Procura um determinado texto por todo o projeto.
- **Fonte de controle:** Realizar o controle de versão do código diretamente pelo Visual Studio Code.
- **Execução e *debug*:** Atalho para executar o código em modo *debug* ou apenas executar o código.
- **Extensões:** Concede acesso a várias extensões disponíveis para serem instaladas no Visual Studio Code.

2.8 Mongo DB

O MongoDB é um banco de dados orientado a documentos que possui código aberto e foi projetado para armazenar uma grande escala de dados, além de permitir que se trabalhe de forma eficiente com grandes volumes (KOVACS, 2022).

Este banco de dados é um modelo *NoSql* e salva seus documentos em um formato *JSON*. O MongoDB pode ser executado tanto através de uma nuvem quanto localmente. Por causa do formato em que o MongoDB salva seus arquivos (*JSON*), os atributos dos documentos ficam mais flexíveis e podem vir a variar até mesmo dentro do próprio modelo, não se prendendo a uma estrutura fixa.

Assim como foi citado pelo (MONGODB, 2022), o MongoDB conta com os seguintes tipos de consultas: ad hoc, índices e agregação , possibilitando consultas em tempo real de uma forma rápida e eficaz. A seguir, estes tipos de consultas são detalhadas.

- **Consultas ad hoc:** A expressão Ad hoc é uma expressão em latim que significa “para este propósito“. Ou seja, a consulta é criada apenas para satisfazer aquela necessidade específica, aquele propósito, em um momento específico (REIS, 2017). Este tipo de consulta é vista com mais frequência em sistemas de *Data Mining* ou *Business Intelligence* onde ocorrem um número maior de solicitações não planejadas.

- **Consultas com índices:** As consultas a partir de índices fazem com que em uma busca, o banco de dados não tenha que percorrer todos os documentos. Os índices são estruturas de dados especiais que armazenam uma pequena porção de um conjunto de dados da coleção em uma forma fácil de percorrer. O índice armazena o valor de um atributo específico ou conjunto de atributos (JUNIOR, 2019). O objetivo é facilitar o processamento da consulta, fazendo com que menos arquivos sejam lidos, menos recursos sejam gastos na busca e que o tempo de busca pelo dado seja o menor possível.

- **Consultas com agregação:** No MongoDB agregações são operações responsáveis por agrupar dados e retornar resultados computados. Esses dados podem ser um conjunto de referenciações entre as tabelas de dados, uma contagem de documentos encontrados ou uma soma de valores entre atributos. Uma agregação é uma relação de um para muitos, onde existe um documento proprietário de outros mas não há dependência entre eles, fazendo com que ambos os documentos existam sem necessariamente haver uma relação estabelecida entre eles.

2.9 GitLab

GitLab é uma plataforma de hospedagem de código-fonte. Ela permite que profissionais de desenvolvimento contribuam em projetos privados ou abertos (mais conhecidos como projetos *open source*) (BERTOLA, 2019).

A plataforma oferece vários recursos similares para hospedagem de código-fonte, sendo alguns como *pull request*[2], revisão de código, edição *inline*, *fork*[3] e clone[4] de repositórios, além de integrações com ferramentas de terceiros(BERTOLA, 2019).

Utilizar a plataforma GitLab para hospedagem de um projeto é fundamental para o controle de versão do código-fonte, uma vez que ela permite que qualquer integrante com permissão para o repositório possa criar novas ramificações do projeto e ir acrescentando mudanças no código principal durante o desenvolvimento do projeto. Essas ramificações permitem que as alterações realizadas por cada desenvolvedor não sejam sobrescritas, garantindo uma melhor organização e progressão para o projeto.

Durante o desenvolvimento de um projeto, não existe um fluxo obrigatório para realizar alterações, uma vez que cada empresa trabalha de forma diferente e contém seus próprios modelos e regras de integração. De maneira geral é possível analisar como é realizado as alterações em um projeto, na Figura 2.6.

Através da Figura 2.6 é possível observar que o fluxo é composto por um ramo que contém a criação de uma *feature*. Essa ramificação é formada a partir do ramo *develop*, que é geralmente definido como um ramo de teste. Logo após a finalização da feature[5], as mudanças são passadas para a *develop*, e por fim são transferidas para o ramo *master*, geralmente definido como a principal ramificação de um projeto.

---

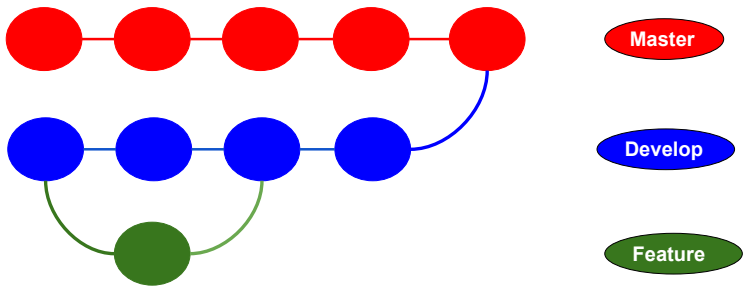
[2] É um mecanismo no qual o desenvolvedor sinaliza que concluiu o desenvolvimento de uma atividade (SILVA, 2022).

[3] É uma cópia de um repositório. Realizar o *fork* em um repositório permite que sejam feitas experiências sem que projeto original seja comprometido (GITHUB, 2022a).

[4] Ao criar um repositório no GitHub.com, ele passa a existir como um repositório remoto. É possível clonar o repositório para criar uma cópia local no seu computador e sincronizar entre os dois locais (GITHUB, 2022b).

[5] São ramos para o desenvolvimento de uma funcionalidade específica, por convenção elas tem o nome iniciado por *feature*, por exemplo: feature/cadastro-usuarios. É Importante ressaltar que esses ramos são criados sempre à partir do ramo *develop* (PANZOLINI, 2018).

Figura 2.6 – Fluxo de Alteração pelo Gitlab



Fonte: Do próprio autor

2.10 One Signal

O One Signal é uma ferramenta que oferece um serviço de envio de notificações automáticas para plataformas *web* ou móveis, conhecidas como notificações *push*.

Notificações *push* são pequenos *pop-ups* que aparecem na tela do usuário, contendo frases ou perguntas que, geralmente, visam engajar o leitor. Um exemplo é quando você recebe mensagens de aplicativos de comida em sua tela de bloqueio (FARIA, 2020).

Além de ser uma ótima forma de engajar um usuário e chamar sua atenção, notificações *push* são acionadas de forma discreta, evitando que o usuário tenha que parar o que está fazendo para remover a notificação de alguma parte da tela.

Segundo (SIGNAL, 2022), o One Signal oferece outros serviços, como : envio de *e-mail*, envio de mensagens e um painel gerenciador, que possibilita um melhor gerenciamento do envio de mensagens e notificações.

Os serviços oferecidos pelo One Singal podem trazer diversas vantagens, tanto para uma aplicação *web* quanto para uma aplicação móvel. Segundo (SIGNAL, 2022), as principais vantagens oferecidas pela plataforma são:

- Rápida integração da plataforma no sistema do cliente;
- Mensagens automatizadas;

- Relatórios em tempo real;
- Alta escalabilidade;
- Entrega inteligente;
- Compara o desempenho de envio entre mensagens;
- Envio personalizado de mensagem para um determinado conjunto de usuários;

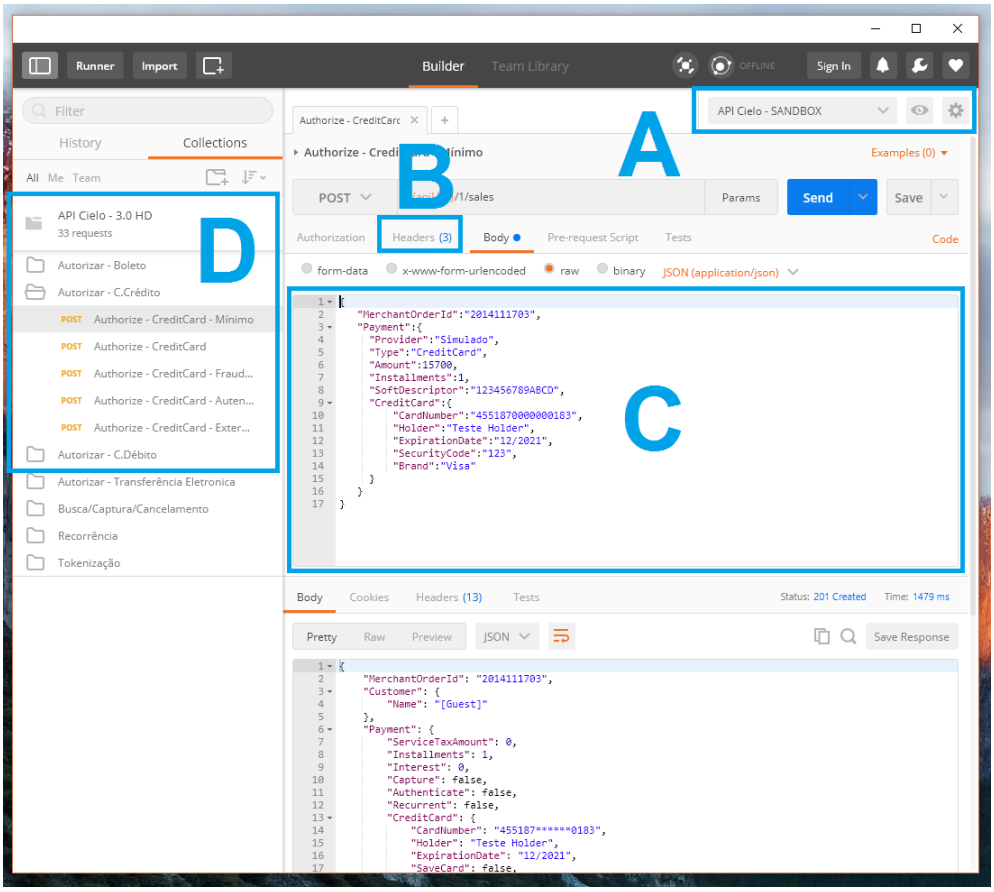
2.11 Postman

O Postman é uma ferramenta que facilita aos desenvolvedores criar, compartilhar, testar e documentar APIs. Isso é feito, permitindo aos usuários criar e salvar solicitações HTTP e HTTPs simples e complexas, bem como ler suas respostas (DESENVOLVEDORES, 2022).

As equipes de desenvolvimento podem utilizar o Postman para armazenar as rotas de cada projeto, facilitando o teste de atividades ou até mesmo servindo como um exemplo do funcionamento das rotas.

Na Figura 2.7 é apresentado a tela inicial da ferramenta Postman e seus componentes.

Figura 2.7 – Componentes da plataforma Postman



Fonte: (DESENVOLVEDORES, 2022)

Segundo (DESENVOLVEDORES, 2022) os componentes apresentados na Figura 2.7, são definidos da seguinte forma:

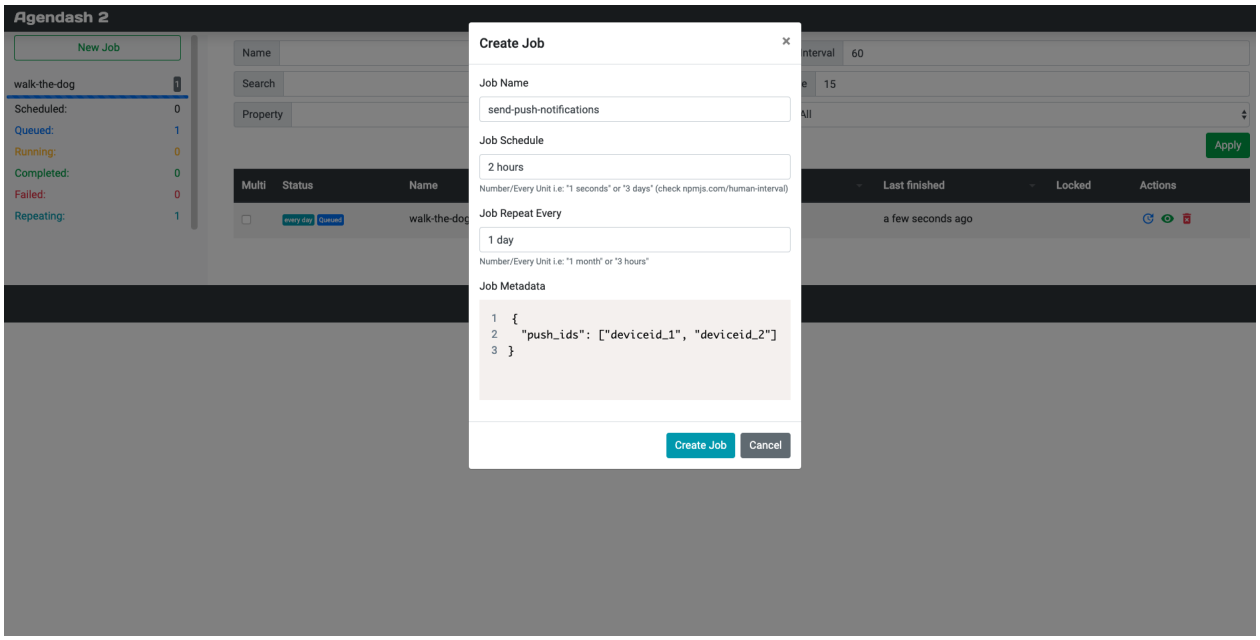
- **Ambiente (Figura (2.7 A)):** Ambiente para onde serão direcionadas as requisições;
- **Header (Figura (2.7 B)):** Campo para armazenar chaves e *tokens* de autenticação para rotas privadas;
- **Corpo da requisição (Figura (2.7 C)):** É o conteúdo enviado junto com as requisições;
- **Coleção (Figura (2.7 D)):** Local que contém todas os exemplos e códigos que podem ser utilizados na API.

2.12 Agenda

É uma biblioteca para o envio automático de eventos; disponibilizando uma maneira prática e rápida de realizar envios programados dentro de um sistema. Esta biblioteca também

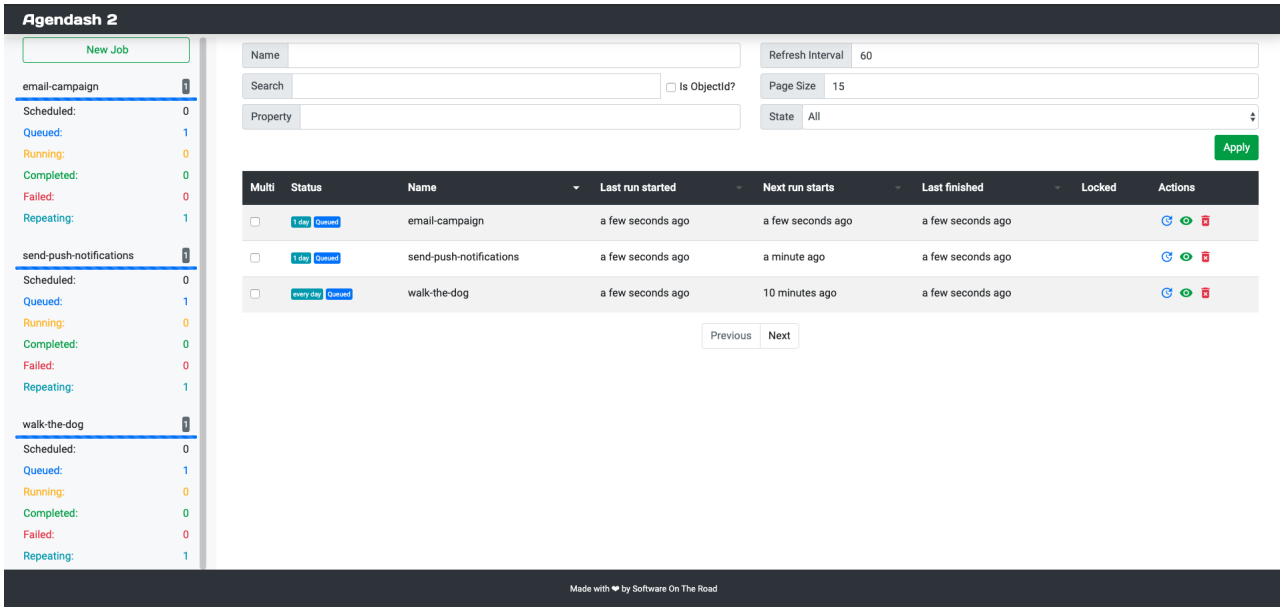
conta com uma interface gráfica, chamada de Agendash. Utilizando esta interface é possível ter um melhor gerenciamento dos eventos, uma vez que seu uso permite a edição, criação e consultas dos eventos adicionados na plataforma, apresentados respectivamente na Figura 2.8 e na Figura 2.9.

Figura 2.8 – Tela de criação e edição de um evento



Fonte: (BOROVIAK, 2022)

Figura 2.9 – Tela de listagem dos eventos



Fonte: (BOROVIAK, 2022)

Algumas das principais vantagens do uso da Agenda segundo (OPENBASE, 2022), são:

- Despesas gerais mínimas, a Agenda visa manter sua base de código pequena;
- Camada de persistência apoiada pelo Mongo DB;
- Programação com prioridade configurável, simultaneidade, repetição e persistência dos resultados do trabalho;
- Fila de trabalho apoiada por eventos em que você pode se conectar.

2.13 Node-cron

O Cron para Node é um pacote NPM(*Node Package Manager*) que nos permite fazer o agendamento de tarefas baseado em uma regra de tempo. Ele é baseado no *Cron* do Linux e seu funcionamento segue a mesma linha. Com ele é possível definir uma função para ser executada de tempos em tempos, ou seja, ela será agendada para ser executada dentro do Node. É uma maneira bastante eficaz para tarefas repetitivas que precisam rodar em segundo plano, como o envio de notificação, *backup* de banco de dados, entre outras (ESCUDELARIO, 2018).

O agendamento de eventos para o Node-cron é escalado em 6 etapas, são elas: segundos, minutos, horas, dia do mês, mês e dia da semana. Para realizar um agendamento com sucesso não é necessário incluir todas as etapas. A etapa de segundos por exemplo, é opcional e caso o usuário não queira especificar os segundos, o evento vai ser executado no primeiro segundo do horário agendado.

Na Figura (2.10), é mostrado um exemplo onde a primeira etapa da configuração do evento e omitida, e a configuração é feita somente usando as outras 5 etapas. Nesta Figura também é possível perceber que as etapas ainda não foram definidas e estão com o valor padrão (*). Quando um evento é agendado com todas as etapas no valor padrão, aquele evento será executado a cada minuto.

Figura 2.10 – Tela de listagem dos eventos

```
var cron = require('node-cron');

cron.schedule('* * * * *', () => {
  console.log('running a task every minute');
});
```

Fonte: (NPM, 2022)

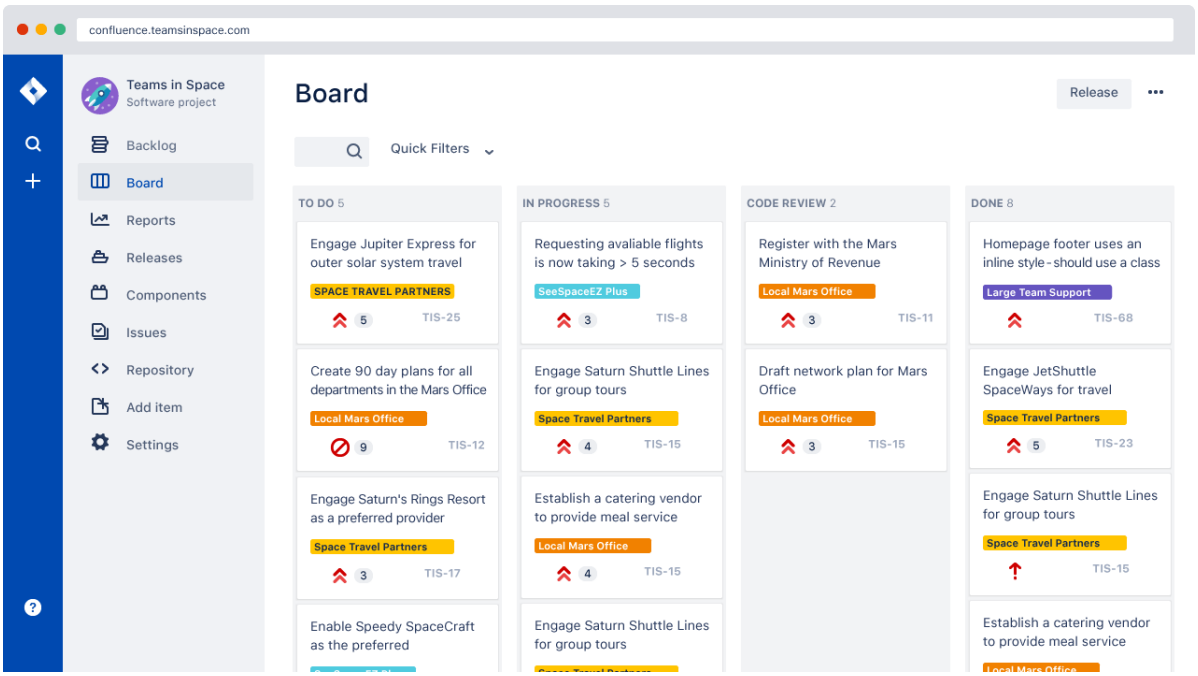
2.14 Jira Software

O Jira Software faz parte de uma família de produtos desenvolvida para ajudar equipes de todos os tipos a gerenciar o trabalho. A princípio, Jira foi desenvolvido como um rastreador de *bugs* e itens. Hoje, o Jira é uma poderosa ferramenta de gerenciamento para todos os tipos de casos de uso, desde gerenciamento de casos de teste e requisitos, até desenvolvimento ágil de *software* (ATLASSIAN, 2022).

Para equipes que praticam metodologias ágeis, o Jira Software oferece painéis *Scrum* e *Kanban* prontos para uso. Os quadros são centros de gerenciamento de tarefas, em que as tarefas são mapeadas para fluxos de trabalho personalizáveis. Os quadros oferecem transparência em todo o trabalho em equipe e visibilidade sobre o status de cada item de trabalho. Os recursos de rastreamento de tempo e os relatórios de desempenho em tempo real (gráficos de *burn-up/down*, relatórios de *sprint*, gráficos de velocidade) permitem que as equipes monitorem de perto a produtividade ao longo do tempo (ATLASSIAN, 2022).

Durante as atividades realizadas, apresentadas neste documento, o Jira software foi utilizado como uma ferramenta para auxiliar no desenvolvimento ágil de *software* e apenas o painel *Scrum* foi utilizado pela equipe de desenvolvimento. Um exemplo do painel *Scrum* é apresentado na Figura(2.11).

Figura 2.11 – Fluxo de Atividade Jira



Fonte: (DRUMOND, 2022a)

2.15 Sendgrid

Sendgrid é um serviço de entrega de *e-mails* que facilita a comunicação com os clientes. Este serviço é muito importante para empresas que usam uma estratégia de *e-mail marketing* em andamento e querem fugir das caixas de *spam* (GESTÃOCLICK, 2022).

Além de ser uma uma ótima ferramenta para *e-mails* de *marketing*, a plataforma possibilita a personalização dos *e-mails* de acordo com as necessidades e objetivos da empresa que utiliza os serviços do Sendgrid. Uma vez que a ferramenta foi integrada no sistema, a organização tem a liberdade de editar o conteúdo dos *e-mails*, assim como escolher os usuários que irão receber os *e-mails* e o horário de envio.

Segundo (GESTÃOCLICK, 2022), as principais vantagens da ferramenta são:

- Infraestrutura escalável (serve tanto para Startups quanto para empresas multinacionais);
- Suporte global;
- Líder no setor em entregabilidade;
- Analítica personalizável e em tempo real.

3 DESENVOLVIMENTO

Neste capítulo será apresentado como as esquipes na empresa Zeester são divididas, o fluxo de desenvolvimento das atividades realizadas pela equipe de desenvolvimento, o ambiente de desenvolvimento e, por fim, as atividades realizadas durante o estágio na empresa Zeester.

3.1 Organização das equipes na Zeester

A empresa Zeester conta com 25 integrantes, que estão divididos em cinco equipes, sendo elas liderança, *design*, desenvolvimento, comunicação e *marketing*. As equipes são descritas a seguir:

- **A equipe de liderança** é encarregada de avaliar os resultados entregues, assim como também é responsável pelas decisões impostas sobre os produtos e serviços oferecidos pela empresa.
- **A equipe de *design*** atua entre a equipe de liderança e desenvolvimento, uma vez que, primeiro, a equipe de liderança impõe requisitos para os modelos de *design* e, em seguida, os protótipos são criados e compartilhados para a equipe de desenvolvimento.
- **A equipe de desenvolvimento** é responsável pela manutenção e implementação de novas funcionalidades de acordo com os requisitos designados.
- **A equipe de *marketing*** é responsável por cuidar das redes sociais da empresa, divulgando lançamentos e aumentando sua notoriedade.
- **A equipe de comunicação** é responsável pelo atendimento dos usuários em caso de problemas, reclamações e dúvidas.

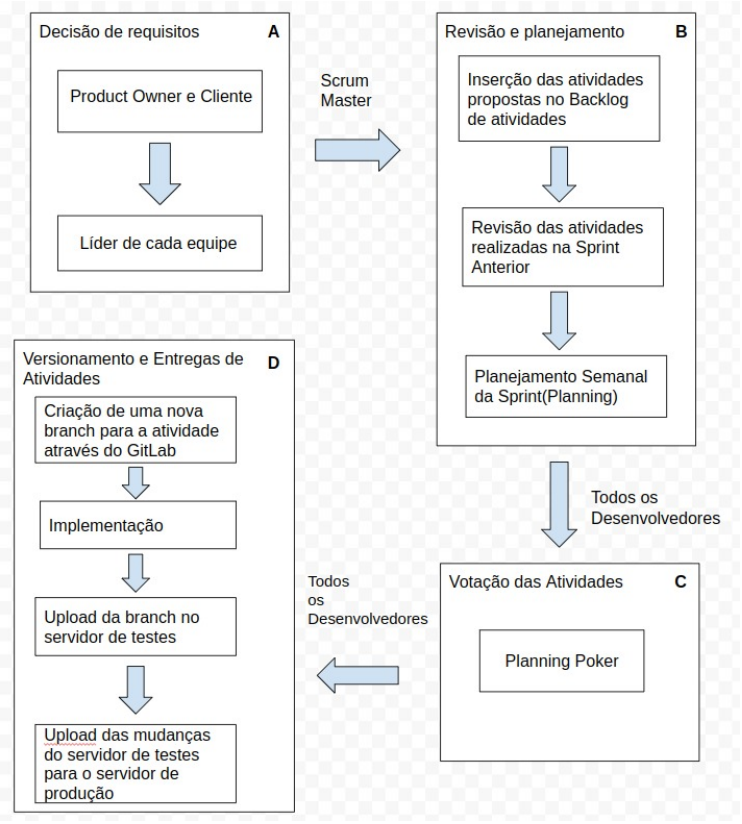
A equipe de desenvolvimento possui, no total, 6 integrantes, os quais estão subdivididos em duas equipes: *back-end* e *front-end*. A primeira é composta por 3 integrantes, sendo dois deles desenvolvedores plenos e um sênior, que por sua vez é o *Scrum master* da equipe de desenvolvimento. Este autor participou da equipe de *back-end* realizando as implementações necessárias para a melhoria e manutenção da plataforma Blesss e seus microsserviços.

3.2 Fluxo de desenvolvimento

Esta sessão é responsável por apresentar as etapas do fluxo de desenvolvimento da equipe *back-end* na empresa Zeester, uma vez que o autor deste documento fora alocado na mesma. Durante a apresentação deste capítulo, também serão citadas quais foram as metodologias e ferramentas utilizadas em cada etapa.

A primeira etapa do fluxo de desenvolvimento (Figura 3.1 A) para novas funcionalidades envolve o diretor da empresa atuando como *Product Owner*. Juntamente com o cliente o diretor vai estabelecer as ações necessárias para atender os requisitos propostos. Logo após a definição das ações necessárias, o líder de cada equipe é notificado, com o objetivo de analisar as novas atividades e determinar quais serão as dificuldades de cada uma.

Figura 3.1 – Fluxo de Desenvolvimento



Fonte: Do próprio autor

Após a finalização da primeira etapa, é utilizada a metodologia *Scrum*, através da plataforma Jira Software. As atividades propostas são inseridas no *Backlog* da plataforma pelo *Srcum Master*, mas caso haja necessidade, qualquer participante pode sugerir alguma outra atividade que vai ajudar no desenvolvimento de novas funcionalidades ou na manutenção de ferramentas que já existam. Assim que todas as atividades foram inseridas no *Backlog*, o *Scrum Master* é

responsável por explicar cada uma e retirar as dúvidas dos participantes. Por fim, as atividades listadas no *Backlog* são utilizadas nas reuniões de planejamento.

A segunda etapa (Figura 3.1 B) consiste no planejamento da *Sprint*, que acontece semanalmente. Esse reunião tem como objetivo revisar as atividades realizadas na semana anterior, buscando analisar se as tarefas foram concluídas, se houve algum imprevisto durante o processo desenvolvimento das atividade e consequentemente resolver qualquer problema que tenha surgido neste processo. Após a revisão, é feita uma avaliação do que melhorou e o que precisa melhorar. Essa avaliação engloba tanto quesitos das tarefas realizas na *Sprint* quanto questões pessoais dos desenvolvedores. Logo após a conclusão da etapa de revisão, o *Scrum Master* da início a *Planning*. Através do *Backlog*, as próximas tarefas da *Sprint* são selecionadas e distribuídas de forma balanceada entre os desenvolvedores.

Na terceira etapa (Figura 3.1 C), acontece a estimativa das atividades selecionadas na etapa anterior, através do conceito de *Planning Poker*. Este conceito faz com que pra cada atividade os desenvolvedores votem uma determinada dificuldade para a realização da atividade. Os votos podem ser valores baixos como 0 ou 1 que significam que a atividade a ser feita é bem simples assim como os valores podem ser mais altos como 3 ou 5, simbolizando uma atividade com um nível de dificuldade mais alto. Em casos extremos existem os votos que são acima de 5, o que significa que provavelmente aquela atividade não será finalizada na *Sprint* que foi criada. Os votos são disponibilizados através sequência de números de Fibonacci (0,1,2,3,5,8,13). Por fim, assim que todos os participantes votam, os valores são coletados e é feito a média para a atribuição do valor de dificuldade da atividade. Em caso da média não ser um número inteiro, a nota sempre será arredondada para o número inteiro mais próximo. Assim que a votação é finalizada, o *Scrum Master* pergunta aos colaboradores se estão de acordo com os pesos votados de cada atividade. Esta verificação da atribuição do peso das atividades é feita com o objetivo de garantir que nenhum desenvolvedor será sobrecarregado ou ficará com falta de atividades durante a *Sprint*.

O controle de versões de todos os projetos é realizado pelo GitLab. Para que um desenvolvedor possa iniciar sua tarefa, o primeiro passo é criar uma ramificação do código a partir da *branch* de desenvolvimento, dando início a última etapa do fluxo de desenvolvimento (Figura 3.1 D). A ramificação de desenvolvimento é responsável por armazenar o código do projeto na versão de testes. Assim que o desenvolvedor termina suas atividades em uma determinada *branch*, é necessário unir o que foi feito para a *branch* de desenvolvimento, para que

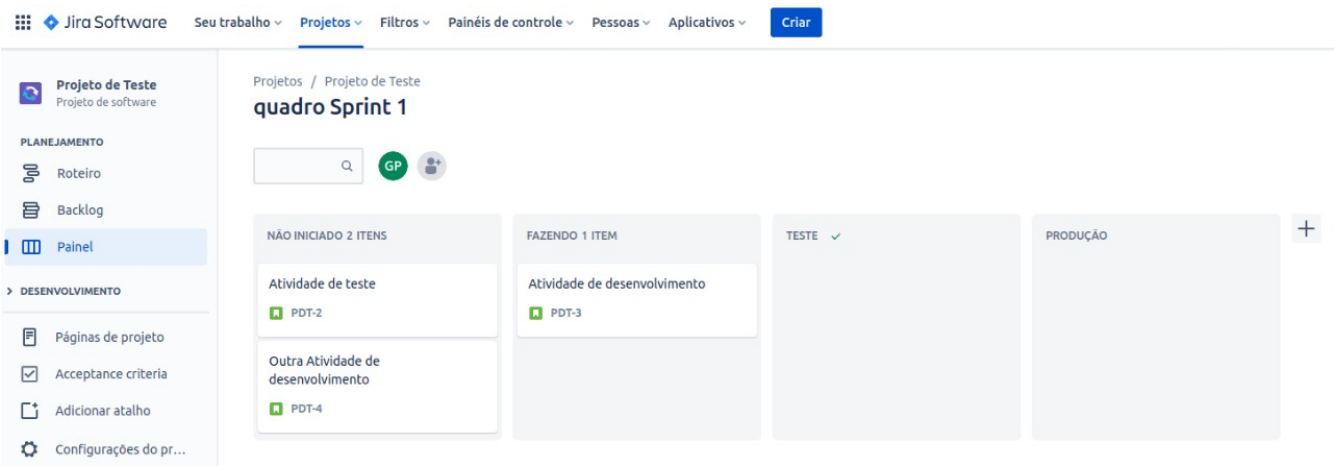
seja possível testar as novas mudanças a partir do servidor de testes. Quando todos os testes são realizados, caso não seja encontrado nenhum problema, as mudanças são atualizadas para a *branch* de produção, a qual é responsável por armazenar o código que é disponibilizado para os clientes.

3.3 Ambiente de Desenvolvimento

Esta sessão é responsável por apresentar o fluxo do ambiente de desenvolvimento durante a realização das atividades na empresa Zeester, citando as tecnologias e os métodos que foram utilizados.

O início de desenvolvimento de qualquer atividade começa com a consulta da mesma na plataforma Jira. Nesta plataforma fica listado quais são as atividades que precisam ser feitas durante a semana e quem é o responsável por cada uma. O primeiro passo é escolher a atividade e, troca-lá para a tabela de atividades em desenvolvimento. Isso é feito para avisar a equipe que a mesma já está sendo desenvolvida, como o exemplo mostrado na Figura 3.2.

Figura 3.2 – Fluxo de Atividade Jira



Fonte: Do próprio autor

Após a escolha da atividade, é necessário acessar o repositório da API na qual atividade escolhida será desenvolvida através do GitLab. Tendo acesso ao repositório, o desenvolvedor é responsável por criar uma nova *branch* a partir da *branch* de desenvolvimento.

Com a *branch* criada, o próximo passo é abrir um editor de código fonte. O editor escolhido pelo time de desenvolvimento foi o Visual Studio Code. É através desta ferramenta que o código de toda tarefa é desenvolvido. Após ter aberto o editor de código, o desenvolvedor deve mudar a *branch* local para a *branch* que foi criada para o desenvolvimento da atividade esco-

lhida. Assim que a *branch* local for alterada, é possível dar início ao processo desenvolvimento da tarefa. Durante este processo o desenvolvedor deve executar a API na qual atividade está sendo desenvolvida para garantir que as mudanças feitas não estejam quebrando nenhum outro funcionamento da API.

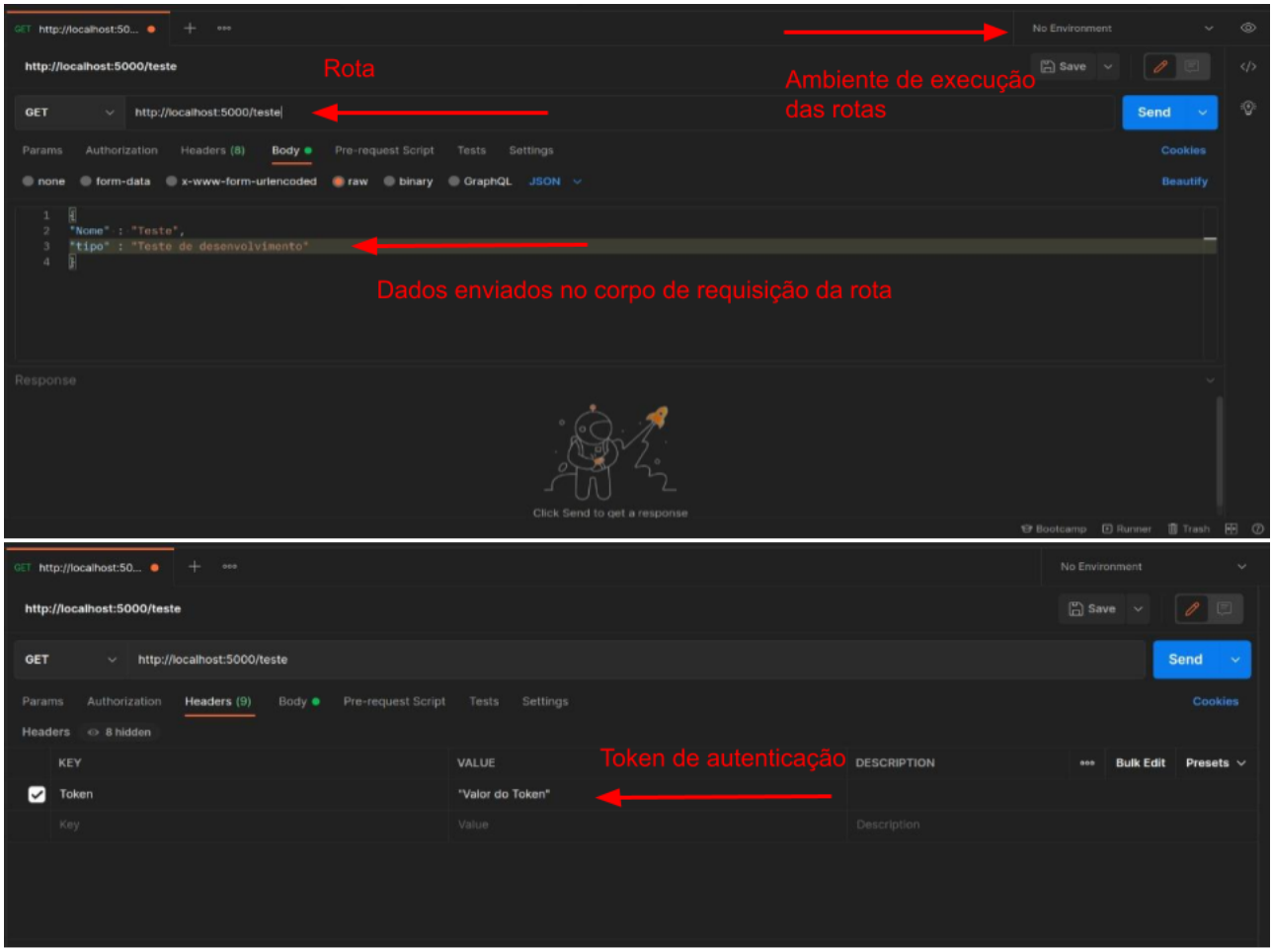
A execução de toda API na empresa Zeester é feita através da ferramenta Docker. É através dela que qualquer desenvolvedor pode executar qualquer API sem se preocupar com a configuração de ambiente. Assim que o desenvolvedor executa o Docker, a API começa a rodar em seu computador, fazendo com que seja possível testar a atividade criada, assim como analisar se o desenvolvimento da mesma afetou qualquer outro comportamento ou função na API.

Para que seja possível testar se alguma atividade feita está funcionando adequadamente no *back-end* de uma API, é necessário chamar as rotas envolvidas. Na Zeester, a ferramenta escolhida entre os desenvolvedores foi o Postman. Para dar incio ao procedimento de teste de uma rota, o desenvolvedor deve abrir o Postman e, antes de chamar qualquer rota, ele deve verificar em qual ambiente aquela rota vai ser chamada. As rotas podem ser chamadas no ambiente local, de teste ou no de Produção. Assim que o ambiente é configurado, é necessário verificar se já existe uma rota para o teste da tarefa a ser testada, caso não haja o desenvolvedor é responsável por criar uma nova rota, assim como preencher os dados necessários para que aquela rota seja executada e testada corretamente, isso envolve *tokens* de autenticação e dados que podem ser passados por parâmetro ou pelo corpo da requisição da rota. Assim que a rota é corretamente configurada, o desenvolvedor começa a executá-la, com o objetivo de analisar os resultados que a mesma vai retornar. O exemplo do fluxo de teste na ferramenta Postman pode ser consultado na Figura 3.3.

Quando uma tarefa passa no processo de testes, ela é enviada para a *branch* de desenvolvimento. Para realizar este procedimento, o desenvolvedor deve abrir o seu editor de código e subir as mudanças locais para o repositório pertencente a atividade realizada. Assim que as mudanças são feitas e enviadas para o repositório, é necessário acessar o GitLab e criar um *merge request* para que as mudanças feitas sejam enviadas para a *branch* de desenvolvimento. Por fim, após o envio das mudanças para a *branch* de desenvolvimento, é necessário fazer um novo *merge request* para enviar as mudanças do ambiente de teste para o ambiente de produção.

Assim que uma atividade entra no ambiente de testes, o desenvolvedor deve acessar a plataforma Jira, e mudar a tarefa que está sendo realizada para o quadro de tarefas em ambiente

Figura 3.3 – Fluxo de Desenvolvimento no Postman



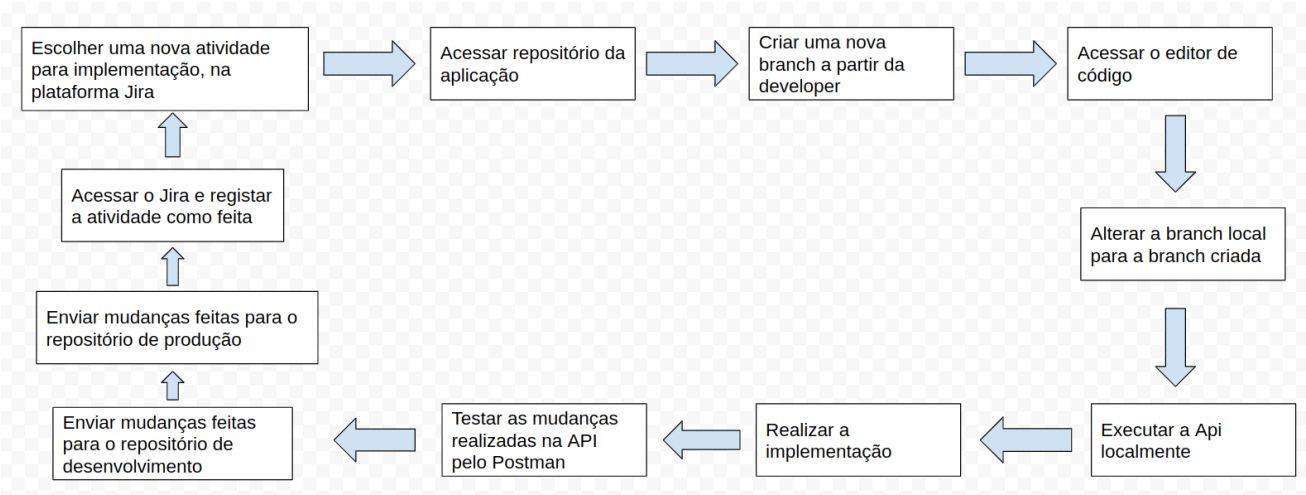
Fonte: Do próprio autor

de teste, e quando uma atividade passa do ambiente de testes para o ambiente de produção ela deve ser enviada para o quadro de tarefas feitas. Assim que um desenvolvedor termina um tarefa, ele está apto a começar uma nova atividade seguindo o fluxo apresentado na Figura 3.4.

3.4 Atividades Realizadas

- **Integração com a API do OneSignal:** Esta atividade teve como objetivo integrar uma API de notificações para o sistema principal da empresa, o Blesss. O primeiro passo da atividade foi criar uma nova coleção no banco de dados que seria responsável por armazenar os novos dados das notificações. Em seguida, foi iniciada a integração do OneSignal no sistema, instalando *tokens* de acesso no ambiente de desenvolvimento para o serviço de notificações, criando funções para programar notificações para usuários específicos em horários controlados, assim como foram desenvolvidas algumas rotas para gerenciar

Figura 3.4 – Fluxo do Ambiente de Desenvolvimento

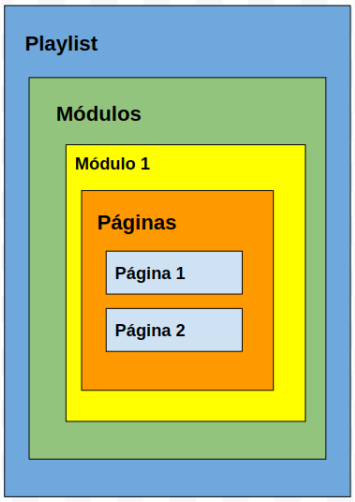


Fonte: Do próprio autor

todo o fluxo de notificações no Blesss. As rotas criadas tinham como objetivo oferecer um CRUD(*Create, read, update, delete*) para o sistema de notificações, sendo possível criar notificações personalizadas, editar notificações existentes, excluir alguma notificação indesejada e listar para cada usuário as notificações recebidas.

- **Refatoração no modelo de *playlists* no banco de dados:** Esta atividade tinha como objetivo melhorar a maneira como as *playlists* armazenavam seus módulos e seus episódios. Antes da refatoração, cada *playlist* armazenava seus módulos que funcionavam como se fossem temporadas dentro de um vetor com todos os dados de cada módulos, assim como em cada módulo era salvo um vetor de episódios contendo todos os dados de cada um. Este modelo de armazenamento pode ser visualizado através da Figura 3.5.

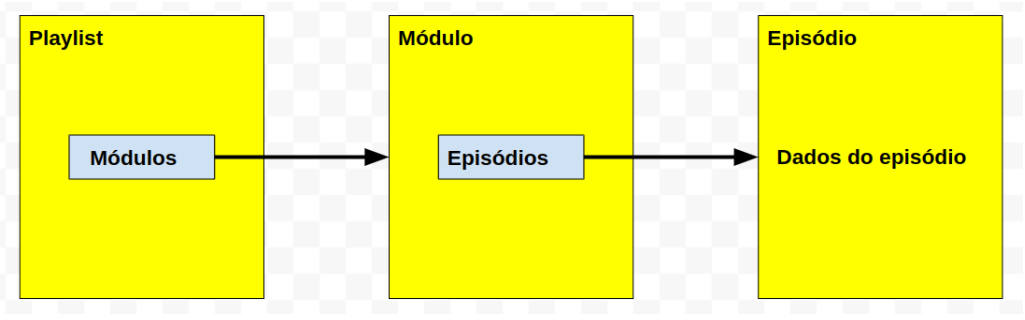
Figura 3.5 – Fluxo de armazenamento da coleção de *playlists* antes da refatoração



Fonte: Do próprio autor

O principal trabalho realizado nesta atividade foi reestruturar este modelo antigo para um modelo mais organizado e que deixasse somente as informações essências de módulos e episódios no modelo de *playlists*. A solução foi criar uma coleção para módulos e uma coleção para episódios, sendo assim, as informações gerais de cada coleção ficariam armazenadas um um local exclusivo somente para aquele tipo de documento. Após a separação das coleções ainda era necessário associar os módulos a sua *playlist*, assim como, os episódios ao seu módulo, então foi feito uma referenciação da coleção de módulos dentro de *playlists* e o mesmo foi feito para os episódios em seus respectivos módulos, criando então um novo fluxo de armazenamento, representado na Figura 3.6

Figura 3.6 – Fluxo de armazenamento da coleção de *playlists* depois da refatoração



Fonte: O próprio autor

- **Implementação do microsserviço de lives:** O Blesss foi desenvolvido como um monolito, sendo assim, existiam vários serviços dentro do mesmo sistema. Esta atividade teve como objetivo retirar o serviço que possibilitava o gerenciamento e transmissão de lives

dentro do sistema do Blesss para uma nova API, através da criação de um microsserviço. Inicialmente, estruturou-se a API que iria comportar o microsserviço, criando um novo repositório e desenvolvendo a base daquela. Por exemplo, o servidor para se comunicar com a API do Blesss e, ao mesmo tempo, estabelecer a conexão do serviços de socket e redis, uma vez que essas tecnologias foram responsáveis por gerenciar os usuários e as salas de cada live. Após a criação da nova API de lives, foi necessário passar toda a implementação que envolvia o gerenciamento e execução de lives dentro do Blesss para API de lives. Assim que toda a implementação de lives foi removida do sistema principal, foi necessário criar rotas de comunicação entre as duas APIS para que o microsserviço de lives fosse disponibilizado para o Blesss. Por fim, foi realizado uma bateria de testes unitários para garantir que todas as funções de lives estavam funcionando corretamente e de maneira estável.

- **Implementação do microsserviço de envio de notificações:** Após o desenvolvimento do serviço de notificações dentro da API do Blesss, foi decidido pela equipe de desenvolvimento passar este serviço para um nova API responsável exclusivamente pela criação e envio das notificações geradas pelo Blesss. Esta atividade teve como objetivo deixar a API do Blesss responsável apenas por armazenar e gerenciar os dados gerados pela criação das notificações, enquanto a nova API ficaria por conta de conter as funções e rotas necessárias por criar e enviar notificações. O primeiro passo desta atividade foi criar um novo repositório para API de notificações e em seguida foi necessário implementar um novo servidor na API de notificações para poder executa-la. Assim que a nova API foi ativada, foi necessário transferir as chaves e *tokens* de acesso do serviço do Onesignal que estavam no Blesss para a API de notificações, uma vez que é essencial ter acesso a esse serviço para conseguir realizar o envio de notificações. Após ter as chaves e *tokens* configurados, todas funções responsáveis por criar e enviar notificações na API do Blesss foram transferidas para API de notificações. Em seguida foi necessário criar rotas na API de notificações para que o Blesss pudesse utilizar os serviços de criação e envio de notificações novamente. Com as rotas criadas, o último passo foi realizar uma bateria de testes unitários para garantir que todas as funções estavam funcionando como deveriam e analisar se a conexão entre as duas APIS estava estável.

- **Automação no envio de notificações programadas:** Após a implementação do microsserviços de notificações, a equipe de desenvolvimento concluiu que o sistema de

automação do envio de notificações não atendia aos padrões de qualidade da empresa, tanto pela biblioteca que estava sendo usada, o node-cron, quanto pelo fato de que essa automação estava sendo feita através da API do Blesss. Assim surgiu a necessidade de desenvolver um microsserviço responsável apenas pelo envio automático de notificações programadas. Esta atividade teve como objetivo remover o cron job de envio de notificações programadas do Blesss e transferir essa responsabilidade para outra API. Para iniciar esta atividade foi necessário criar um novo repositório e desenvolver um servidor na nova API criada. Com a API ativada, o próximo passo foi escolher uma nova biblioteca para gerenciar e realizar o envio automático das notificações programadas. A biblioteca escolhida foi a Agendash, e em seguida foi integrada na nova API. Com a nova biblioteca integrada foi necessário estabelecer uma conexão com a API de envio de notificações, uma vez que era ela a responsável pelo serviço do envio de notificações. Em seguida houve a implementação do envio automático para as notificações programadas. Assim que todas as funções foram implementadas na nova API, foi necessário estabelecer uma conexão entre a API criada nesta atividade e a API do Blesss, uma vez que era necessário saber para quais usuários as notificações iriam ser enviadas e a API do Blesss ficou então responsável por mandar uma rota com uma lista de usuários que estavam qualificados para receber uma notificação programada. Quando foi possível obter a lista de usuários, o processo para enviar uma notificação programada estava completo. Sendo assim foi necessário executar alguns testes para garantir que o envio das notificações estavam acontecendo na hora que elas foram programadas para serem enviadas e se a conexão entre as APIS estava estável. Por fim, assim que todos os testes foram realizados, o cron job responsável pelas notificações programadas no Blesss foi removido, dando fim a esta atividade.

- **Implementar o envio de mensagens automáticas:** Esta atividade foi realiza em um sistema interno da empresa Zeester, que administrava as doações feitas pelos clientes. Estes clientes eram, então, categorizados como mantenedores. A atividade realizada tinha como objetivo enviar uma mensagem para o celular e para o *e-mail* de um mantenedor assim que ele realizasse uma doação. Para realizar a atividade foi necessário implementar uma função que chamasse um evento para o envio de uma mensagem. Este evento era disponibilizado pela ferramenta Sendgrid, que por sua vez, foi a tecnologia utilizada para realizar o envio automático mensagens. Esta ferramenta estava integrada em um outro

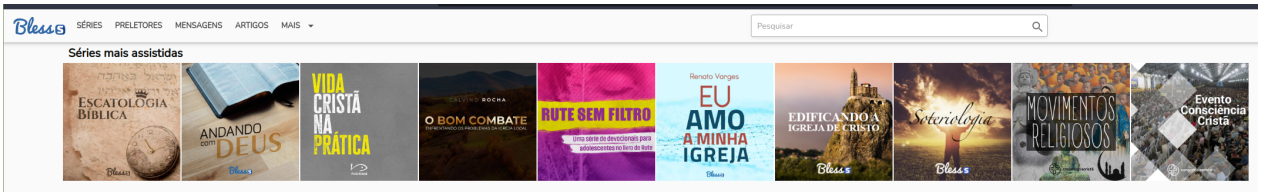
sistema interno da Zeester, responsável pelas funções de comunicação. O Sistema de comunicações já estava conectado com o API de mantenedores, descartando a necessidade de configurar uma conexão entre APIS. Na fase de desenvolvimento foi apenas necessário implementar o evento de envio de mensagens, disponibilizada pela ferramenta. Após a implementação do evento, a função foi utilizada na rota chamada quando uma doação era realizada. A rota quando acionada reconhecia o usuário e consultava o *e-mail* e número de telefone do usuário para realizar o envio da mensagem, automatizando o envio de menagens no sistema.

- **Implementação de novos status para as lives transmitidas no sistema:** A API do Blesss utiliza um microsserviço para poder realizar a transmissão e gerenciamento das lives dentro do sistema e, nesta atividade, foi decidido que seria necessário criar novos status paras as lives realizadas, com o objetivo de ajudar os usuários a entender o status de uma live com mais detalhes. Antes da realização desta atividade, os status disponíveis indicavam somente se uma live estava aberta ou não. Com a implementação dos novos status foi possível notificar aos usuários o progresso das lives, tais como o início e término da transmissão, momento de postagem do conteúdo e a opção de gravar a live no sistema. A realização desta atividade envolveu a alteração do status de uma live no banco de dados durante a sua transmissão, uma vez que o sistema tinha que estar apto a trocar o status de uma live automaticamente de acordo com o andamento da live. Na fase de implementação desta atividade foi necessário atualizar as condições de status que eram utilizadas nas lives durante sua transmissão e também foi necessário editar os eventos lançado pelo socket, uma vez que ela era a ferramenta utilizada por alterar o status das lives em tempo real. Por fim, foi necessário alterar a coleção das lives no banco de dados para poderem aceitar os novos status utilizados e serem salvas corretamente. As lives que já tinham sido finalizadas antes desta atividade tiveram seus status alterados através de uma realização de um *script*, que atualizada seu status como finalizada para as lives que não continham gravação no sistema e as que eram gravadas recebiam um status de video, indicando que havia um conteúdo disponível no site. A alteração da coleção foi realizada somente na API do Blesss, que por sua vez era responsável por armazenar o conteúdo dos vídeos enquanto as alterações de status em tempo real e edição dos eventos acionados pelo socket foram realizados na API responsável pelo microsserviços de lives.

- **Implementação de tratamento de erros no microsserviço de lives:** Nesta atividade foi implementado uma estrutura de tratamento de erro nas funções do microsserviços de lives. O desenvolvimento desta atividade envolve todas as funções disponibilizadas no sistema em um bloco *try catch*, para que o erro fosse tratado de uma forma específica e, depois, ele era enviado para uma camada superior até chegar na camada de rotas onde ele era retornado como resposta, uma vez que ele já foi tratado nas camadas anteriores. O objetivo desta atividade foi tratar os erros para que os mesmos não travassem a API e ajudar os usuários e desenvolvedores a entender o retorno de uma função, seja para debugar ou para um usuário entender onde está errando. No fim da implementação dos blocos de tratamento de erro, foi realizado uma bateria de testes unitários para garantir que todos os erros foram tratados da forma correta e estavam sendo acionados.

- **Implementando função para retornar as *playlists* mais assistidas:** Esta atividade foi realizada na API do Blesss e teve como objetivo implementar uma nova função para obter as *playlists* mais assistidas da plataforma. A necessidade da implementação desta atividade surgiu pelo fato de que a página inicial do site Blesss não estava engajando tanto os usuários, e acabava não influenciando os mesmos a consumirem o conteúdo disponibilizado na plataforma. Por conta destes fatores, foi implementado então, na página principal, um carrossel indicando quais eram as *playlists* mais assistidas, servindo como um gatilho para que os usuários assistissem mais conteúdos e permanecessem na plataforma. Para a implementação desta atividade foi desenvolvida uma função responsável por acessar a coleção de *playlists* no banco de dados e fazer uma contagem das 10 *playlists* mais assistidas na plataforma. Após o desenvolvimento da função, a mesma foi usada para retornar os dados presentes no carrossel de *playlists*, assim como mostra a Figura 3.7

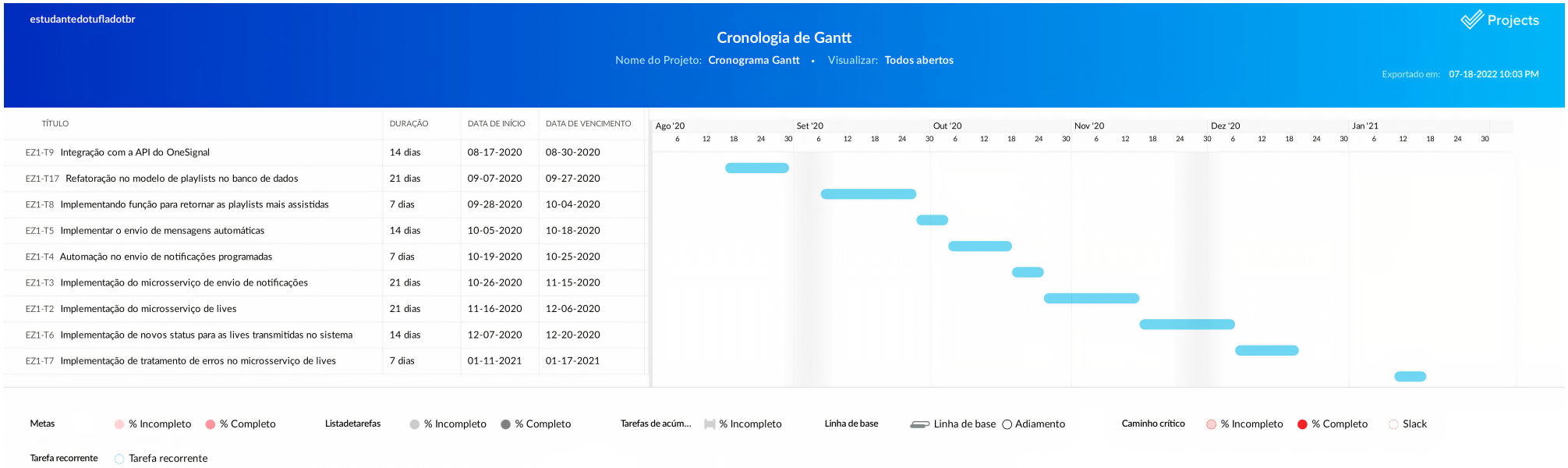
Figura 3.7 – Carrosel das playlists mais assitidas na plataforma do Blesss



Fonte: (BLESSS, 2022)

3.5 Cronograma

Esta sessão tem como objetivo apresentar as atividades realizadas durante o tempo de estágio na empresa Zeester no time de desenvolvimento *back-end*.



Fonte: Do próprio autor

4 CONCLUSÃO

O objetivo deste trabalho foi apresentar as atividades realizadas pelo autor deste documento, como desenvolvedor *back-end* na empresa Zeester. Durante o desenvolvimento das atividades, foi possível agregar conhecimento em tecnologias e ferramentas utilizadas na área de desenvolvimento web, tais como: a linguagem Java Script, Docker e banco de dados. Conhecimentos sobre arquiteturas de desenvolvimento também foram obtidos através da integração de microsserviços de envio automático de notificações *push* pelo One Signal, envio automático de mensagens pelo SendGrid e agendamento automático de eventos através da plataforma Agenda.

Algumas das atividades realizadas envolveram conteúdos apresentados em aulas oferecidas pelo curso de Sistemas de Informação pela UFLA, no qual o autor deste documento está matriculado. A matéria Estrutura de dados foi muito importante para o aprendizado de boas práticas de programação e manipulação de estrutura de dados, como, árvores, listas e vetores. A matéria de Banco de dados também ofereceu uma ótima base de conhecimento, uma vez que durante as atividades realizadas pelo o autor, se mostrou necessário o conhecimento prévio de manipulação de dados pertinentes, através do banco de dados não relacional Mongo DB. Por fim, a matéria de Segurança, Auditoria e Avaliação de Sistemas de Informação, ofereceu conhecimento sobre como desenvolver estruturas de forma segura. O conhecimento em Segurança de *Software* foi fundamental para o desenvolvimento das atividades que envolviam a manipulação de dados sensíveis de pessoas reais. Nesse contexto, a plataforma do Blesss foi desenvolvida respeitando a confidencialidade de *tokens* garantindo rotas seguras e não comprometedoras.

Durante o tempo de estágio como desenvolvedor na empresa Zeester, o autor analisou algumas sugestões que poderiam melhorar o desempenho do desenvolvimento geral da equipe, tais como: documentação das APIS responsáveis pela plataforma Blesss e documentação para os modelos no banco de dados. A plataforma Blesss, apresenta um código extenso e uma arquitetura complexa envolvendo microsserviços. A documentação das APIS iria auxiliar na explicação das funcionalidades e rotas presentes nas mesmas, facilitando na adaptação de novos desenvolvedores para a empresa, e também iriam ajudar os desenvolvedores mais experientes da empresa a lembrar das funcionalidades mais específicas ou pouco utilizadas no projeto. A falta de documentação para o banco de dados prejudica o desempenho dos desenvolvedores recém chegados na empresa, uma vez que a plataforma Blesss trabalha com uma estrutura complexa de banco de dados. A falta de documentação desta estrutura gera um atraso no desenvolvimento

de tarefas que envolvem a manipulação de dados pertinentes e faz com que os desenvolvedores se confundam durante a realização de atividades deste gênero.

REFERÊNCIAS

AMAZON. **O que é o Docker ?** 2022. Disponível em: <https://aws.amazon.com/pt/docker/>. Acesso em: 01 ago. 2022.

AMAZON. **O que é uma API ?** 2022. Disponível em: <https://aws.amazon.com/pt/what-is/api/>. Acesso em: 30 jul. 2022.

ARTINE, D. **Docker: Desvendando o DockerFile**. 2019. Disponível em: <https://www.alura.com.br/artigos/desvendando-o-dockerfile>. Acesso em: 17 ago. 2022.

ATLASSIAN. **Para que serve o Jira?** 2022. Disponível em: <https://www.atlassian.com/br/software/jira/guides/use-cases/what-is-jira-used-for#Jira-for-requirements-&-test-case-management>. Acesso em: 12 ago. 2022.

BERTOLA, F. **Git, Github e Gitlab: o que são e principais diferenças**. 2019. Disponível em: <https://www.zup.com.br/blog/git-github-e-gitlab>. Acesso em: 01 ago. 2022.

BLESSS. **Séries mais assistidas**. 2022. Disponível em: <https://blesss.org/meublesss>. Acesso em: 17 ago. 2022.

BOROVIAK koresar V. **Agendash**. 2022. Disponível em: <https://github.com/agenda/agendash>. Acesso em: 11 ago. 2022.

CONSULTORIA, C. **Scrum, a arte de fazer o dobro na metade do tempo!** 2019. Disponível em: <https://cromoconsultoria.com.br/scrum-a-arte-de-fazer-o-dobro-na-metado-do-tempo/>. Acesso em: 11 ago. 2022.

DESENVOLVEDORES, C. **Postman Cielo**. 2022. Disponível em: <https://developercielo.github.io/tutorial/postman>. Acesso em: 11 ago. 2022.

DEVMEDIA. **Tecnologia Node.js**. 2022. Disponível em: <https://www.devmedia.com.br/guia/node-js/40312>. Acesso em: 30 jul. 2022.

DRUMOND, C. **Gerenciamento de projetos no agile**. 2022. Disponível em: <https://www.atlassian.com/br/agile/project-management>. Acesso em: 21 set. 2022.

DRUMOND, M. **Scrum**. 2022. Disponível em: <https://www.atlassian.com/br/agile/scrum>. Acesso em: 02 ago. 2022.

EDUCAÇÃO, X. Scrum: O que é, como funciona e por que você precisa aprender. 2022. Disponível em: <https://blog.xpeducacao.com.br/scrum/>. Acesso em: 01 ago. 2022.

ESCUDELARIO, B. D. F. **Agendando tarefas com o Cron para Node**. 2018. Disponível em: <https://openbase.com/js/agenda>. Acesso em: 12 ago. 2022.

ESPINHA, R. G. **Backlog do produto: entenda o que é e como fazer nos seus projetos**. 2022. Disponível em: <https://artia.com/blog/backlog-do-produto/>. Acesso em: 11 ago. 2022.

EXPRESS. **Express**. 2017. Disponível em: <https://expressjs.com/>. Acesso em: 30 jul. 2022.

FARIA, D. **Notificações push: o que são e como se beneficiar**. 2020. Disponível em: <https://www.assoweb.com.br/notificacoes-push-o-que-sao-e-como-se-beneficiar/>. Acesso em: 11 ago. 2022.

GESTÃOCLICK. **SendGrid: o que é e como funciona**. 2022. Disponível em: <https://gestaoclick.com.br/blog/integracao-sendgrid-mail/>. Acesso em: 12 ago. 2022.

GITHUB. **Bifurcar um repo**. 2022. Disponível em: <https://docs.github.com/pt/get-started/quickstart/fork-a-repo>. Acesso em: 22 ago. 2022.

GITHUB. **Clonar um repositório**. 2022. Disponível em: <https://docs.github.com/pt/repositories/creating-and-managing-repositories/cloning-a-repository>. Acesso em: 22 ago. 2022.

HANASHIRO, A. **VS Code - O que é e por que você deve usar?** 2021. Disponível em: <https://www.treinaweb.com.br/blog/vs-code-o-que-e-e-por-que-voce-deve-usar>. Acesso em: 11 ago. 2022.

HAT, R. Api rest. **O que é API REST?**, 2020. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>. Acesso em: 30 jul. 2022.

JUNIOR, J. R. F. Índices em mongodb. 2019. Disponível em: <https://www.linkedin.com/pulse/%C3%ADndices-em-mongodb-jose-r-f-junior/?originalSubdomain=pt>. Acesso em: 01 ago. 2022.

KOVACS, L. **O que é e para que serve o MongoDB?** 2022. Disponível em: <https://tecnoblog.net/responde/o-que-e-e-para-que-serve-o-mongodb>. Acesso em: 22 ago. 2022.

L, A. **O Que é JavaScript e Como Funciona**. 2019. Disponível em: <https://www.weblink.com.br/blog/programacao/o-que-e-javascript/>. Acesso em: 11 ago. 2022.

LENON. **Node.js – O que é, como funciona e quais as vantagens**. 2018. Disponível em: <https://www.opus-software.com.br/node-js/>. Acesso em: 11 ago. 2022.

MICROSOFT. **Introdução**. 2022. Disponível em: <https://code.visualstudio.com/docs>. Acesso em: 01 ago. 2022.

MONGODB, I. **O que é o MongoDB?** 2022. Disponível em: <https://www.mongodb.com/pt-br/what-is-mongodb>. Acesso em: 01 ago. 2022.

NPM. **node-cron**. 2022. Disponível em: <https://www.npmjs.com/package/node-cron>. Acesso em: 12 ago. 2022.

OPENBASE. **Agenda**. 2022. Disponível em: <https://openbase.com/js/agenda>. Acesso em: 11 ago. 2022.

PANZOLINI, B. **Gerenciando seus branches com o Git Flow**. 2018. Disponível em: <https://tableless.com.br/git-flow-introducao>. Acesso em: 22 ago. 2022.

REIS, F. dos. O que é uma consulta ad hoc em bancos de dados. 2017. Disponível em: <http://www.bosontreinamentos.com.br/bancos-de-dados/o-que-e-uma-consulta-ad-hoc-em-bancos-de-dados>. Acesso em: 01 ago. 2022.

SIGNAL, O. 2022. Disponível em: <https://onesignal.com/>. Acesso em: 11 ago. 2022.

SILVA, G. **O que é Pull Request (PR)?** 2022. Disponível em: <https://coodesh.com/blog/dicionario/o-que-e-pull-request-pr/>. Acesso em: 22 ago. 2022.

SOUZA, I. de. Javascript: o que é, como funciona e por que usá-lo no seu site. **Desenvolvimento para web**, 2019. Disponível em: <https://rockcontent.com/br/blog/javascript/>. Acesso em: 30 jul. 2022.

TECNISYS. **QUAL É A DIFERENÇA ENTRE UMA IMAGEM DOCKER E UM CONTAINER?** 2019. Disponível em: <http://www.tecnisys.com.br/noticias/2021/qual-e-a-diferenca-entre-uma-imagem-docker-e-um-container>. Acesso em: 17 ago. 2022.

W3TECHS. **Technology Brief**. 2022. Disponível em: <https://w3techs.com/technologies/details/cp-javascript/>. Acesso em: 30 jul. 2022.