INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA

WANDREUS MÜHL DOURADO

DESENVOLVIMENTO DE API PARA PLATAFORMA DE EVENTOS DO IFRO

Vilhena - RO
2021

WANDREUS MÜHL DOURADO

DESENVOLVIMENTO DE API PARA PLATAFORMA DE EVENTOS DO IFRO

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – campus Vilhena, realizado em cumprimento de requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Gilberto Pereira da Silva

Vilhena - RO

2021

FICHA CATALOGRÁFICA

Biblioteca IFRO – Campus Vilhena

D739d

DOURADO, Wandreus Mühl

Desenvolvimento de API para plataforma de eventos do IFRO / Wandreus Mühl Dourado – Vilhena, Rondônia, 2021.

63f. ; il.

Orientador Prof. Esp. Gilberto Pereira da Silva

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

1. API 2. Eventos 3. Kanban I. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO II. Título

006.7882

Bibliotecária responsável Rosilene Maria do Couto Marques CRB 11/321

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 11107804
Rodovia BR 174, KM 3, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

ATA DE DEFESA DE MONOGRAFIA

Na data 15/12/2021 realizou-se a sessão pública de defesa da Monografia intitulada **Desenvolvimento de API para plataforma de eventos do IFRO** apresentada pelo aluno **Wandreus Muhl Dourado (2019105053013-4)** do Curso **Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (Vilhena)**. Os trabalhos foram iniciados às **19:00** pelo Professor **Gilberto Pereira da Silva** presidente da banca examinadora, constituída pelos seguintes membros:

- **Gilberto Pereira da Silva** (Orientador)
- **Flavio de Almeida Andrade Lico** (Examinador Interno)
- **Marco Antonio Augusto de Andrade** (Examinador Interno)

A banca examinadora, tendo terminado a apresentação do conteúdo da Monografia, passou à arguição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] APROVADO **Nota: 97**

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Gilberto Pereira da Silva** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

VILHENA / RO, 15/12/2021

Documento assinado eletronicamente por **Wandreus Muhl Dourado**, Discente, em 20/12/2021, às 09:01, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gilberto Pereira da Silva**, Orientador, em 17/12/2021, às 16:14, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Flavio de Almeida Andrade Lico**, Examinador Interno, em 17/12/2021, às 17:24, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marco Antonio Augusto de Andrade**, Examinador Interno, em 19/12/2021, às 09:49, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Este trabalho é dedicado à minha família, de sangue e de consideração,
a qual me ajudou e apoiou para que eu chegasse até aqui

Agradecimentos

À minha família que tanto me apoiou durante esse trajeto. Aos meus pais Antonio e Marinilde, por acreditarem em mim, me incentivarem e me darem as condições necessárias para que isso acontecesse. Aos meus irmãos Rafaela e Bruno, pelo companheirismo, amizade, carinho e ensinamentos durante toda a vida.

Ao professor Gilberto Pereira da Silva, por aceitar orientar-me durante o processo de desenvolvimento desse trabalho e pelos ensinamentos, apoio e confiança durante toda a minha formação. E aos professores Juliano Fischer Naves e Marco Antonio Augusto de Andrade, por toda a ajuda que me ofereceram, além de todos os ensinamentos. Sinto-me privilegiado de ter tido dois coorientadores.

À todos do corpo docente do meu curso que, de alguma forma, contribuíram para o meu aprendizado.

À equipe de que trabalhou ao meu lado durante o processo de desenvolvimento deste *software*. Obrigado pela ajuda e pelo companheirismo Emily Lucas e Rebecca Duarte.

Aos meus "irmãos de outra mãe": Ericky, pela amizade, companheirismo e colaboração durante todo o curso; Mateus, pelo apoio, companheirismo e amizade desde o ensino médio e Pábulo, um dos responsáveis por eu ter escolhido este curso, obrigado por sua amizade e companheirismo que muito me ajudaram durante minha trajetória desde o ensino médio.

E aos demais colegas e amigos que contribuíram em diversos momentos, em especial à Ariane, por todas as dicas, colaboração e incentivos durante a produção dessa monografia.

O afã por descobrir alimenta a criatividade em todos os campos, não só na ciência. Se chegássemos à meta, o espírito humano se murcharia e morreria.

Stephen Hawking

Resumo

Eventos escolares são métodos muito úteis para que alunos absorvam novos conhecimentos, além de que, quando bem divulgados, promovem um retorno positivo à instituição de ensino, seja por melhorar a forma como a sociedade a enxerga ou por facilitar o surgimento de novas parcerias. Esse tipo de evento é muito importante, tal qual seu gerenciamento. Esta monografia apresenta o processo de desenvolvimento de parte de uma API (*Application Programming Interface* - Interface de Programação de Aplicações) para plataforma de eventos do IFRO (*Instituto Federal de Educação, Ciência e Tecnologia de Rondônia*). O objetivo do projeto é iniciar o desenvolvimento de um núcleo que forneça as informações referentes aos eventos da instituição. Durante o processo, a metodologia ágil Kanban foi utilizada. Resultando em um MVP com funcionalidades voltadas à criação de eventos e suas atividades, bem como as inscrições que nelas pode-se realizar. A principal contribuição do trabalho é, de fato, iniciar o desenvolvimento desse *software*, o qual continuará em desenvolvimento, a fim de que, uma vez finalizado, possa realizar o gerenciamento dos eventos escolares.

Palavras-chave: API; Plataforma de Eventos; Kanban.

Abstract

School events are very useful methods for students to absorb new knowledge, besides that, when well publicized, they promote a positive return to the educational institution, either by improving the way society sees it or by facilitating the emergence of new partnerships. This type of event is very important, as is its management. This monograph presents the process of developing an API (Application Programming Interface) for IFRO (Federal Institute of Education, Science and Technology of Rondônia) event platform. The purpose of this project is start the development of a nucleus that provides information regarding the events of the institution. During the process, the Kanban agile methodology was used. Resulting in an MVP with features aimed at creating events and their activities, as well as registrations that in them it can be realized. The main contribution of the work is, in fact, initiate the development of this software, which will continue in development, so that, once finalized, it will be able to manage school events.

Keywords: API; Event platform; Kanban.

Lista de ilustrações

Figura 1 – Docker Compose 31
Figura 2 – Modelo Lógico 32
Figura 3 – Diagrama de classes 34
Figura 4 – Diagrama de atividade - criação de evento 35
Figura 5 – Diagrama de sequência - cadastro de usuário 35
Figura 6 – Arquitetura 37
Figura 7 – *Lead Time* - Setembro 39
Figura 8 – *Lead Time* - Outubro 40
Figura 9 – *Lead Time* - Novembro 40
Figura 10 – *Cycle Time* - Setembro 41
Figura 11 – *Cycle Time* - Outubro 41
Figura 12 – *Cycle Time* - Novembro 42
Figura 13 – *Throughput* 43
Figura 14 – Teste automatizado - *Login* 45
Figura 15 – Tags Swagger 46
Figura 16 – Rota DELETE 47
Figura 17 – Rota GET 47
Figura 18 – Rota POST 48
Figura 19 – Rota PUT 48
Figura 20 – Rota "*/evento*" 48
Figura 21 – Seeder - Usuários 49
Figura 22 – Validação de atividades 51
Figura 23 – Requisição - POST "/evento" 53
Figura 24 – Resposta - POST "/evento" - Erro 53
Figura 25 – Resposta - GET "/evento" - Sucesso 54
Figura 26 – *Query* - GET "/evento" 54
Figura 27 – Resposta - GET "/evento/:evento_id/atividade" - Sucesso 55
Figura 28 – Requisição - POST "/evento/:evento_id/atividade/:atividade_id/inscritos" - Sucesso 56
Figura 29 – Resposta - GET "/usuario/:usuario_id/certificado" 57

Lista de tabelas

Tabela 1 – Requisitos funcionais 36
Tabela 2 – Requisitos não funcionais 36
Tabela 3 – Testes e retornos esperados 45

Lista de abreviaturas e siglas

API	Application Programming Interface
CPU	Central Process Unit
CRUD	Acrônimo para Create (criação), Read (consulta), Update (atualização) e Delete (destruição) de dados
HTTP	HyperText Transfer Protocol
ID	Identificador
IFRO	Instituto Federal de Educação, Ciência e Tecnologia de Rondônia
JS	JavaScript
JSON	JavaScript Object Notation
ML	Modelo Lógico
MVP	Most Value Product
ORM	Object-Relational Mapping
RF	Requisitos Funcionais
RQ	Requisitos Não Funcionais
SGBD	Sistema Gerenciador de Banco de Dados
SO	Sistema Operacional
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
VM	Virtual Machine
YAML	YAML Ain't Markup Language

Sumário

1 INTRODUÇÃO 21
1.1 Contexto e problema 21
1.2 Objetivos 21
1.2.1 Objetivo geral 21
1.2.2 Objetivos específicos 21
1.3 Justificativa 21

2 FUNDAMENTAÇÃO TEÓRICA 23
2.1 Interface de Programação de Aplicações 23
2.2 Linguagem de Programação JavaScript 23
2.2.1 Ambiente de execução 24
2.3 Conteinerização 24
2.4 Banco de dados 25
2.5 Versionamento 25
2.5.1 Sistemas centralizados 25
2.5.2 Sistemas distribuídos 26
2.6 Metodologia ágil 26
2.7 Trabalhos similares 27

3 MATERIAIS E MÉTODOS 29
3.1 Materiais 29
3.1.1 Persistência de dados 30
3.2 Métodos 32
3.2.1 Kanban 32
3.2.2 Versionamento 33
3.2.3 Modelagem 33
3.3 Requisitos 35
3.4 Arquitetura do software 36
3.5 Licença de uso 37

4 RESULTADOS E DISCUSSÕES 39
4.1 Gerenciamento de tarefas - Métricas 39
4.1.1 Lead Time e Cycle Time 39
4.1.2 Taxa de transferência 42
4.2 Relatório de testes 43
4.2.1 Testes manuais 43

4.2.2 Testes automatizados 44
4.3 Documentação 46
4.4 Populando o banco 49
4.4.1 Validação de entrada 50
4.5 Implantação 52
4.6 Demonstração do *software* 52

5 CONSIDERAÇÕES FINAIS 59
5.1 Trabalhos futuros 59

REFERÊNCIAS 61

6 LICENÇA MIT 63

1 Introdução

1.1 Contexto e problema

Visando facilitar a organização de eventos internos, o IFRO, por meio da FSLab, está desenvolvendo uma plataforma de eventos. A plataforma será utilizada para o gerenciamento de eventos internos, permitindo que administradores criem e encerrem eventos, tais quais as suas atividades. Futuramente, a plataforma também possibilitará o envio e avaliação de trabalhos científicos por meio da mesma.

O problema que este trabalho busca auxiliar a resolver é: o IFRO não possui uma plataforma de eventos, o que acarreta custos tanto para o gerenciamento de eventos, quanto para a emissão de certificados de participação em atividades.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver parte da API da plataforma de eventos do IFRO, facilitando o gerenciamento dos eventos que ocorrerão na instituição.

1.2.2 Objetivos específicos

Os objetivos específicos são os passos para se atingir o objetivo geral. Os objetivos específicos deste trabalhos são:

- Entender o funcionamento dos eventos institucionais;
- Escolher as tecnologias a serem utilizadas;
- Realizar o levantamento de requisitos funcionais e não funcionais;
- Analisar os requisitos levantados, a fim de identificar as características necessárias para o sistema;
- Disponibilizar a aplicação em ambiente de produção para verificar o seu comportamento.

1.3 Justificativa

Tendo em vista que eventos institucionais são importantes, tanto para a instituição, quanto para os estudantes - que utilizam os certificados de participação como horas complementares,

uma plataforma própria que pode moldar-se a necessidade da instituição é de grande valor. Este trabalho inicia o processo de criação dessa plataforma, desenvolvendo uma parte da API que alimentará a mesma. Além disso, com a API pronta e funcionando, o desenvolvimento de um aplicativo mobile da Plataforma de Eventos se torna mais fácil. Um aplicativo agregaria valor à plataforma e tornaria o acesso à mesma mais prático.

2 Fundamentação teórica

2.1 Interface de Programação de Aplicações

Uma API (*Application Programming Interface*) consiste em um conjunto de regras e características existentes em uma aplicação, os quais possibilitam interações com a mesma através de *softwares* - o que a difere de uma interface humana (MOZILLA, 2021a).

Essas interfaces permitem que a aplicação se comunique com outros serviços e produtos, não necessitando ciência da forma que foram implementados. Esse fato simplifica o desenvolvimento de aplicações, economizando tempo e custos de produção. Tanto no desenvolvimento de novas ferramentas, quanto no gerenciamento das já existentes, o uso de APIs acarreta em uma maior flexibilidade, simplificação de administração e oportunidades de inovação.

As APIs são vistas como contratos, com documentações que representam o acordo entre as duas partes. A estrutura de uma solicitação remota oriunda de uma das partes determinará como a outra parte responderá (RED HAT, 2017).

O uso dessas interfaces de programação simplificam o processo de integração de novos produtos a uma arquitetura preexistente. Fato que pode ser determinante em um cenário de concorrência entre empresas, tendo em vista que nesses ambientes, as necessidades empresariais mudam constantemente para atender os requisitos do mercado.

APIs públicas, como a que foi desenvolvida, simplificam e ampliam as formas de conexão com parceiros. Com elas, libera-se o acesso aos recursos da aplicação sem abrir mão da segurança e controle dos dados. O que é desenvolvido é que determina como essa distribuição será feita e quem terá acesso à ela. Com um bom gerenciamento, a segurança das APIs é potencializada. É possível conectar diversas APIs, assim como criar aplicações que são abastecidas por elas, usando uma integração distribuída que ligue todos os elementos. Essa abertura estimula o desenvolvimento de um ecossistema baseado na API inicial.

2.2 Linguagem de Programação JavaScript

A linguagem de programação JavaScript (comumente abreviado para JS) é uma linguagem interpretada, de alto nível e não tipada, mais conhecida como a linguagem de *scripts* para páginas Web (MOZILLA, 2021b).

Dinâmica e multi-paradigma, a linguagem é conveniente para programações funcionais e orientadas a objeto. De acordo com (FLANAGAN, 2013), sua sintaxe deriva da linguagem Java, das funções de primeira classe da linguagem Scheme e da herança baseada em protótipos da

linguagem Self. Por mais que seja amplamente conhecida como linguagem de *script*, JavaScript é uma linguagem muito mais robusta, eficiente e de uso geral.

2.2.1 Ambiente de execução

Para ter utilidade, toda linguagem deve ter uma plataforma, ou uma biblioteca, ou uma API de funções para que as coisas possam ser executadas. A linguagem JavaScript fornece uma API mínima para trabalhar com *strings*, *arrays*, datas e expressões regulares, mas não inclui funcionalidade de entrada ou saída. Funcionalidades de entrada e saída são responsabilidade do ambiente que hospeda o código. Geralmente, em um cenário *client-side*, esse ambiente é representado por um navegador Web, entretanto, como dito anteriormente, JavaScript não é mais somente uma linguagem de *scripts*, e durante o desenvolvimento desse produto, ela foi utilizada no *server-side*. Para que isso seja possível, é necessário implementar um ambiente de execução da linguagem (FLANAGAN, 2013).

Para esse projeto, utilizou-se o interpretador Node.js. Por se tratar de um ambiente assíncrono de execução de JavaScript orientado a objetos, o Node.js é planejado para o desenvolvimento de aplicações escaláveis de rede. Devido ao fato de ser um ambiente de execução assíncrono, o Node consegue controlar diversas conexões ao mesmo tempo.

Essa abordagem diverge do modelo onde utiliza-se *threads* do SO, o qual é ineficiente e de difícil uso. Além disso, utilizando-se do Node.js, não há necessidade de se preocupar com o congelamento do código. Quase nenhuma função no Node.js realiza operações de entrada e saída, assim sendo, o processo não é bloqueado. Por não existirem operações bloqueantes, sistemas escaláveis são razoavelmente fáceis de serem desenvolvidos (NODE.JS, 2021).

O ambiente também é projetado para que haja uma alta taxa de fluxo e uma baixa latência. Tornando-o um ótima escolha para servir de ambiente de execução de uma biblioteca Web ou de um *framework*.

2.3 Conteinerização

Visando uma melhor escalabilidade e organização estrutural do sistema, o mesmo foi desenvolvido utilizando a arquitetura de microsserviços. Para hospedar cada um desses serviços, foram utilizados contêineres.

Contêineres são pacotes de *software* trazidos da Internet que contêm todas as dependências para o mesmo ser executado. Comparados a VMs, os contêineres compartilham o *kernel* do SO em vez de replicá-lo completamente. Mesmo que hospedar microsserviços em VMs seja possível, essa abordagem é menos viável do que a utilização de contêineres, já que eles ocupam menos espaço e são inicializados mais rapidamente (DOCKER, 2018).

Para permitir a conteinerização da aplicação, utilizou-se o Docker - um *software open-*

source voltado para o desenvolvimento, implementação e gerenciamento de aplicativos conteinerizados.

Docker consiste em um kit de ferramentas que facilita o processo de gerenciamento de contêineres através de simples comandos, ocasionando em automatização e economia de trabalho. Mesmo que a tecnologia de contêineres tenha surgido décadas antes de 2013 - ano em que ocorreu a liberação pública do Docker - "contêineres" e "Docker" são utilizados como sinônimos (IBM, 2021).

2.4 Banco de dados

O MariaDB Server é um dos servidores de bancos de dados mais populares do mundo. Desenvolvido pela equipe de desenvolvimento original do MySQL, o MariaDB converte dados em informações estruturadas. Originalmente projetado como substituto aprimorado do MySQL, o *software* é utilizado devido a sua velocidade, escalabilidade e robustez. Além disso, conta com um amplo ecossistema de mecanismos, *plug-ins* e outras ferramentas que o tornam altamente versátil. MariaDB é um gerenciador de bancos de dados relacionais *open-source* que fornece uma interface SQL para realizar o acesso aos dados. As versões mais recentes do SGBD oferecem recursos para o armazenamento de JSON (MARIADB, 2021).

2.5 Versionamento

Geralmente utilizados ao decorrer do desenvolvimento de *softwares*, sistemas de controle de versão são aplicações que facilitam o gerenciamento de diferentes versões durante o desenvolvimento de algum produto.

O controle de versão funciona com base em um repositório que armazena os arquivos de um projeto. Neste repositório, fica salvo o histórico de versões dos arquivos em questão. Dessa forma, os desenvolvedores conseguem acessar a última versão disponível e realizar uma cópia local, assim podem trabalhar a partir dela, continuando o processo de desenvolvimento. A cada alteração feita, é possível enviar a cópia local ao servidor, atualizando a versão remota a partir das alterações feitas por toda a equipe de desenvolvimento (LACERDA, 2012).

Por meio desses sistemas também é possível reverter para um estado anterior arquivos específicos ou, até mesmo, o repositório completo; comparar as diferenças entre esses estados, além de rastrear os autores dessas alterações.

2.5.1 Sistemas centralizados

Esses sistemas, trabalham com apenas um repositório central para diversas áreas de trabalho. Por ser centralizado, todas as áreas de trabalho precisam primeiramente passar pelo

servidor central, só aí podem se comunicar. Esta é uma abordagem que funciona muito bem com equipes de desenvolvimento de menor porte e que trabalham em uma rede local. Entretanto, a ideia de um repositório central apresenta um ponto de falha. Caso haja problema com o mesmo, todo o sistema para de funcionar.

2.5.2 Sistemas distribuídos

O sistema é composto por vários repositórios independentes entre si. Cada repositório funciona como um backup completo de todos os dados, assim, caso um dos servidores apresente falhas, qualquer um dos outros repositórios pode ser utilizados para a restauração, descartando, a principio, a ideia de haver um repositório mais importante que o outro. Essa abordagem se encaixa muito bem em equipes que atuam em lugares diferentes, já que permite a utilização de distintos fluxos de trabalho, o que não é possível em sistemas centralizados. (CHACON; STRAUB, 2014)

2.6 Metodologia ágil

É impossível falar sobre metodologias ágeis aplicadas ao desenvolvimento de *softwares* sem retornar a 2001, quando um grupo de dezessete desenvolvedores, interessados em descobrir melhores maneiras para a realização do processo de desenvolvimento, se reuniram para declarar alguns princípios que ficariam conhecidos como Manifesto Ágil.

O manifesto conta com quatro valores e doze princípios. Sendo os valores:

- **Indivíduos e interações** mais que processos e ferramentas;
- **Software em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação de contratos;
- **Responder a mudanças** mais que seguir um plano.

A utilização do texto em negrito remete a ideia de que, por mais que haja valores nos itens à direita, deve haver uma maior valorização nos itens à esquerda (BECK et al., 2001).

O Manifesto Ágil serviu como base para as metodologias ágeis. Entre elas, o XP - EXtreme Programming, o Scrum e o Kanban.

O processo de desenvolvimento dessa aplicação foi baseado na metodologia Kanban, a qual foi originada do Sistema Toyota de Produção, no final da década de 1940. Nesse período, a Toyota introduziu o *just in time* como método de produção. Diferentemente do método criado por Henry Ford - o qual consistia em produzir produtos em massa, para depois enviá-los ao

mercado; o *just in time* representa um sistema puxado, ou seja, a produção é baseada na demanda do mercado, evitando desperdícios e poupando tempo (KANBANIZE, 2021).

Kanban é uma palavra oriunda do japonês e significa literalmente cartão de sinal. Geralmente, o quadro Kanban possui três colunas - “Pedido”, “Em progresso” e “Concluído”. Quando bem gerenciado, serve como central de informações em tempo real, evidenciando os gargalos no processo e empecilhos às práticas de trabalho. A metodologia Kanban é fundamental para o processo de melhoria continua (*kaizen*) utilizado na Toyota, sendo o motor que faz esse sistema funcionar .

Para a aplicação da metodologia no desenvolvimento de softwares, utiliza-se um sistema Kanban virtual para limitar o trabalho que está sendo executado. Divergindo do significado da palavra, os cartões utilizados nesses sistemas, não são realmente sinais para puxar mais trabalho. Em vez disso, eles representam o que está sendo trabalhado. O sinal para puxar um novo trabalho é deduzido com base na quantidade visual dos trabalhos em execução subtraída de algum limite.

O uso dessa metodologia limita o trabalho em execução, definindo a capacidade e equilibrando a demanda sobre a equipe de desenvolvimento em relação ao rendimento das entregas. Isso acarreta em um ritmo sustentável de desenvolvimento, facilitando o equilíbrio entre trabalho e vida pessoal da equipe. Apenas o ato de limitar a demanda sobre uma equipe é suficiente para incentivar uma maior qualidade e melhor desempenho. O fluxo aperfeiçoado combinado com a melhor qualidade do produto ajuda na redução dos prazos de entrega, estabelecendo uma cadência de entrega regular e consistente (ANDERSON, 2011).

2.7 Trabalhos similares

Durante a pesquisa, as seguintes plataformas para gerenciamento de eventos foram encontradas:

- EVEN3: plataforma de divulgação de e gerenciamento de eventos. A Even3 era a plataforma utilizada pela instituição. A mesma conta com ótimas ferramentas para o gerenciamento de eventos e emissão de certificados de participação, mesmo para os eventos gratuitos. Entretanto, para eventos gratuitos - os quais a instituição criava - o certificado só fica disponível gratuitamente por 90 dias, após esse período, é cobrado um valor de R$ 9,90 pela emissão de cada certificado. Isso inviabilizou o uso da plataforma, tendo em vista que os certificados são importantes para a formação acadêmica, já que contabilizam as horas complementares.

- SYMPLA: plataforma para a divulgação e gerenciamento de eventos. A Sympla conta com um catálogo extenso de cursos das mais diversas categorias, o que a torna uma boa opção de uso. Outro ponto forte é que para eventos gratuitos, a plataforma não cobra nenhum tipo de taxa, mesmo oferecendo todas as ferramentas de qualidade - as quais atendem a

maioria dos eventos. Entretanto, para os eventos com ingressos pagos, a cobrança de 10% é apenas sobre ingresso vendido (ou o valor mínimo de R$ 2,50)

- EVENTIZE!: plataforma digital para a organização de eventos. A eventize! é uma plataforma bem completa que fornece várias ferramentas, desde a criação e organização de eventos, até a emissão dos certificados de participação. Entretanto, grande parte dessas ferramentas não estão disponíveis para o plano gratuito, além disso, nesse plano, as inscrições são limitadas.

- DOITY: plataforma de organização de eventos. Uma plataforma muito interessante, pois o plano gratuito oferece muitas funcionalidades. Entretanto, além de limitar o número de participantes - o número é bem alto: 3000, a plataforma informa que as inscrições em atividades e o envio de trabalhos científicos, funcionalidades fundamentais para a instituição, devem ser consultadas com a plataforma.

3 Materiais e métodos

Neste capítulo são descritos os materiais e métodos utilizados para o desenvolvimento da solução proposta.

3.1 Materiais

Para o desenvolvimento do produto utilizou-se a linguagem de programação JavaScript[1]. O principal fator que levou a escolher essa linguagem foi a padronização, tendo em vista que futuramente o front-end da aplicação será desenvolvido com um *framework* JavaScript - React.JS[2]. Utilizando-se da mesma linguagem em todas as camadas da aplicação, todo o processo se torna muito mais prático, já que podemos receber os dados no navegador com JavaScript, enviá-los para o back-end e, com o auxílio de uma ORM (*Object-Relational Mapping* - Mapeamento Objeto-Relacional), podemos fazer a manipulação de dados e armazená-los em um banco ainda com a mesma linguagem. Certamente, o fluxo reverso seria, da mesma forma, prático.

Para possibilitar a utilização de JavaScript no *server-side* precisa-se de um ambiente de execução. Para isso, utilizou-se o Node.JS, sendo esse o principal *software open-source* para a execução de comandos JavaScript. Outro fator que influenciou a escolha dessa ferramenta é a sua flexibilidade. Através do NPM (*Node Package Manager* - Gerenciador de Pacotes do Node)[3], têm-se acesso ao maior repositório de pacotes do mundo, fazendo do Node.JS uma plataforma com potencial de utilização em qualquer situação. Essa vasta biblioteca também possibilita a reutilização de códigos, os quais podem ser instalados e utilizados gratuitamente.

Como dito antes, as possibilidades de aplicação ao se utilizar o Node.JS são incontáveis. Mas, para tarefas de desenvolvimento e implantação web, o principal pacote utilizado é o *framework* Express[4]. Sendo ele o *framework* Node mais popular, também é requisito para diversos outros pacotes Node.

Para manipulação de dados foi utilizado o ORM Sequelize[5]. Sequelize é um ORM baseado em *promise* voltado para os bancos de dados: Postgres, MySQL, MariaDB, SQLite e Microsoft SQL Server. Foi uma ferramenta essencial para o desenvolvimento dessa aplicação, pois ele torna as consultas ao banco de dados muito mais simples utilizando-se funções assíncronas. Além disso, com a criação de *models* (modelos de tabelas definidos de acordo

[1] Disponível em https://www.javascript.com
[2] Disponível em https://pt-br.reactjs.org
[3] Disponível em https://www.npmjs.com
[4] Disponível em https://expressjs.com
[5] Disponível em https://sequelize.org

com o ORM), é possível sincronizá-los com o banco de dados - processo que cria todas as tabelas e relacionamentos de maneira automática. Outro ponto a se destacar, é que por não ser baseado em *queries* de SQL puro, caso seja necessário mudar o serviço de banco de dados, não há necessidade de reescrita de código, basta mudar o dialeto do banco de dados no arquivo de configuração do ORM.

3.1.1 Persistência de dados

Para o armazenamento de dados - parte essencial da maioria dos sistemas - foi utilizada a técnica de *conteinerização* do banco de dados. Essa abordagem está sendo cada vez mais utilizada atualmente, tendo em vista que, além de trazer maior flexibilidade e agilidade para o sistema, a utilização de contêineres facilita o escalonamento da aplicação, já que, como dito anteriormente, suas imagens são inicializadas muito mais rapidamente que máquinas virtuais.

Para criação e gerenciamento dos contêineres, utilizou-se o *software open-source* Docker[6]. Juntamente com o Docker, foi utilizado o Docker Compose, uma ferramenta para definir e iniciar multi-contêineres a partir de um arquivo YAML (*YAML Ain't Markup Language* - YAML não é linguagem de marcação), chamado *docker-compose*.

A figura 1 mostra o arquivo *docker-compose* utilizado para esta aplicação.

[6] Disponível em https://www.docker.com

api-plataforma-de-eventos-ifro - docker-compose.yaml

```yaml
version: "3.1"
services:
  mariadb:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_PASSWORD}
      MYSQL_DATABASE: ${DB_NAME}
    ports:
      - ${DB_PORT}:${DB_PORT}
    networks:
      - mariadb-compose-network

  adminer:
    image: adminer
    ports:
      - 8080:8080
    networks:
      - mariadb-compose-network

networks:
  mariadb-compose-network:
    driver: bridge
```

Figura 1 – Docker Compose

Como servidor de banco de dados, a aplicação utiliza o MariaDB[7], sendo esse um dos principais servidores para bancos de dados relacionais. O motivo que levou à escolha do MariaDB como servidor foi o melhor desempenho em relação ao MySQL.

Para facilitar o gerenciamento do conteúdo do banco de dados, foi utilizada uma imagem do Adminer, um gerenciador de banco de dados. A utilização desse software foi de grande ajuda durante a fase de desenvolvimento, pois com ele, o processo de acesso ao banco foi simplificado, não necessitando conectar-se ao *contêiner* do banco de dados e controlá-lo via terminal.

Para permitir a comunicação entre os dois *contêineres*, foi criada uma rede em modo *bridge*.

A princípio, a plataforma seria desenvolvida utilizando um banco de dados orientado a documentos - *MongoDB*. Entretanto, após analisar os dados que deveriam ser armazenados pela aplicação, verifiquei que, para um melhor funcionamento do sistema, utilizar um modelo

[7] Disponível em https://mariadb.org

relacional seria mais vantajoso, tendo em vista a quantidade de relacionamentos que foram levantados. Após isso, levei essa opção ao *product owner*, para que o mesmo analisasse a viabilidade dessa abordagem. Tendo verificado que a minha análise estava correta, ficou decidido que seria melhor a utilização de um banco de dados relacional.

Após essa decisão, foi desenvolvido o ML (Modelo Lógico) conforme a figura 2.

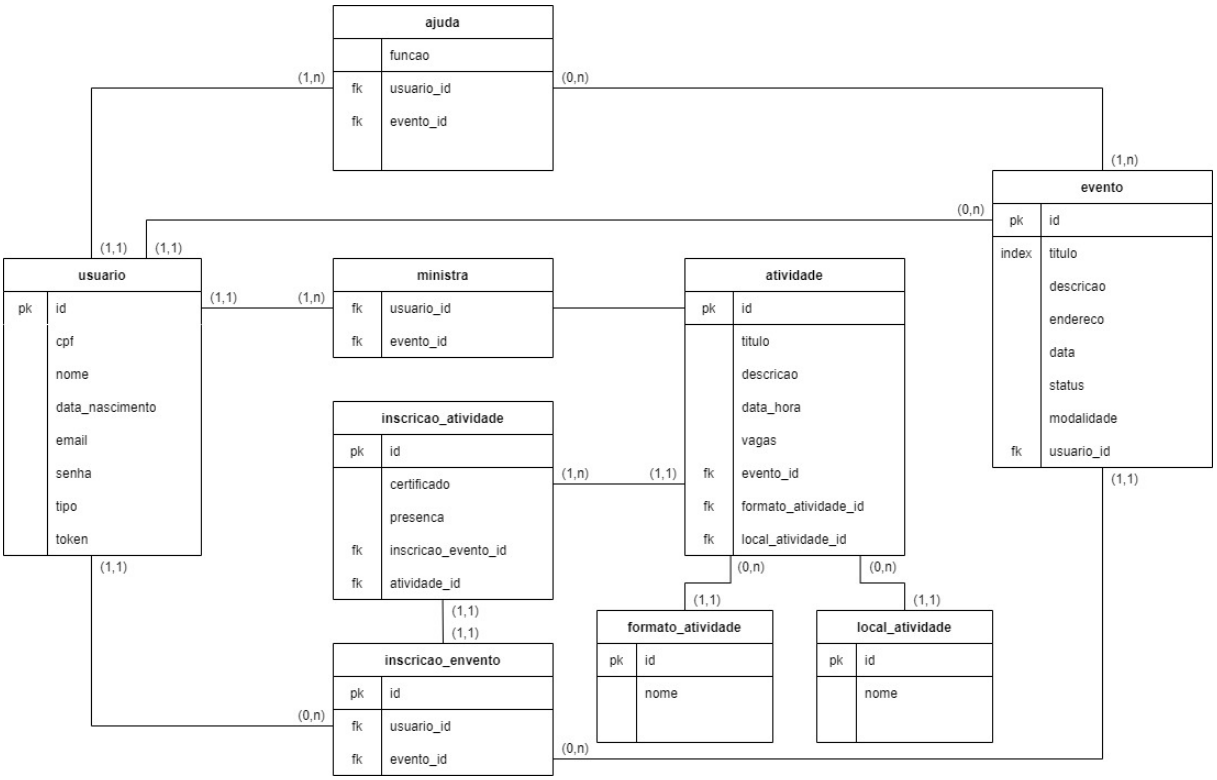

Figura 2 – Modelo Lógico

3.2 Métodos

3.2.1 Kanban

Durante o desenvolvimento do produto, utilizou-se a metodologia Kanban, já citada anteriormente. Para todo o processo, foram utilizados 96 *cards*, os quais representam pequenos pedaços do processo de desenvolvimento. Esses *cards* possuem *tags* que estampam a natureza da atividade. Entre as *tags* mais utilizadas temos:

- Documentation: *cards* que representam atividades relacionadas à documentação da aplicação;
- Enhancement: esses *cards* significam o aprimoramento de uma funcionalidade já desenvolvida;

- Feature: *cards* marcados com essa *tag* significam a implementação de uma nova funcionalidade na aplicação;
- Studies: *tag* que representa atividades relacionadas ao aprendizado de uma nova ferramenta.

O processo de utilização da metodologia foi não apresentou dificuldades, ao contrário, facilitou a gestão do processo de desenvolvimento, além ter sido uma oportunidade para o aprimoramento do uso da metodologia no contexto do desenvolvimento de software, tendo em vista que já havia utilizado essa metodologia em outros momentos.

3.2.2 Versionamento

Para realizar o versionamento do código-fonte utilizou-se um repositório no GitLab da Fábrica de Software da instituição[8]. Nesse repositório foram criadas diversas *branches* (ramificações), cada uma para a resolução de uma atividade do quadro *Kanban*. Após concluir o desenvolvimento e enviar o código para o repositório remoto, é necessário realizar um *merge request* - processo que mescla duas *branches*. Após esse procedimento, a *branch* em questão é excluída. As *merges* eram realizadas na *branch* 'development', e todas as novas ramificações eram criadas a partir dela.

A utilização de um repositório remoto foi de grande importância para o desenvolvimento da solução, pois com ele tornou-se possível rastrear o andamento do desenvolvimento. Além disso, a colaboração é facilitada, já que diversas pessoas podem colaborar em um projeto a partir de um repositório central.

3.2.3 Modelagem

Durante o processo de documentação do projeto, alguns diagramas UML foram criados. Dentre eles o diagrama de classes, demonstrado na figura 3, responsável por modelar a estrutura e relacionamentos das classes que serviram como modelos para a criação de objetos.

[8] Disponível em https://gitlab.fslab.dev/fslab/api-plataforma-de-eventos-ifro

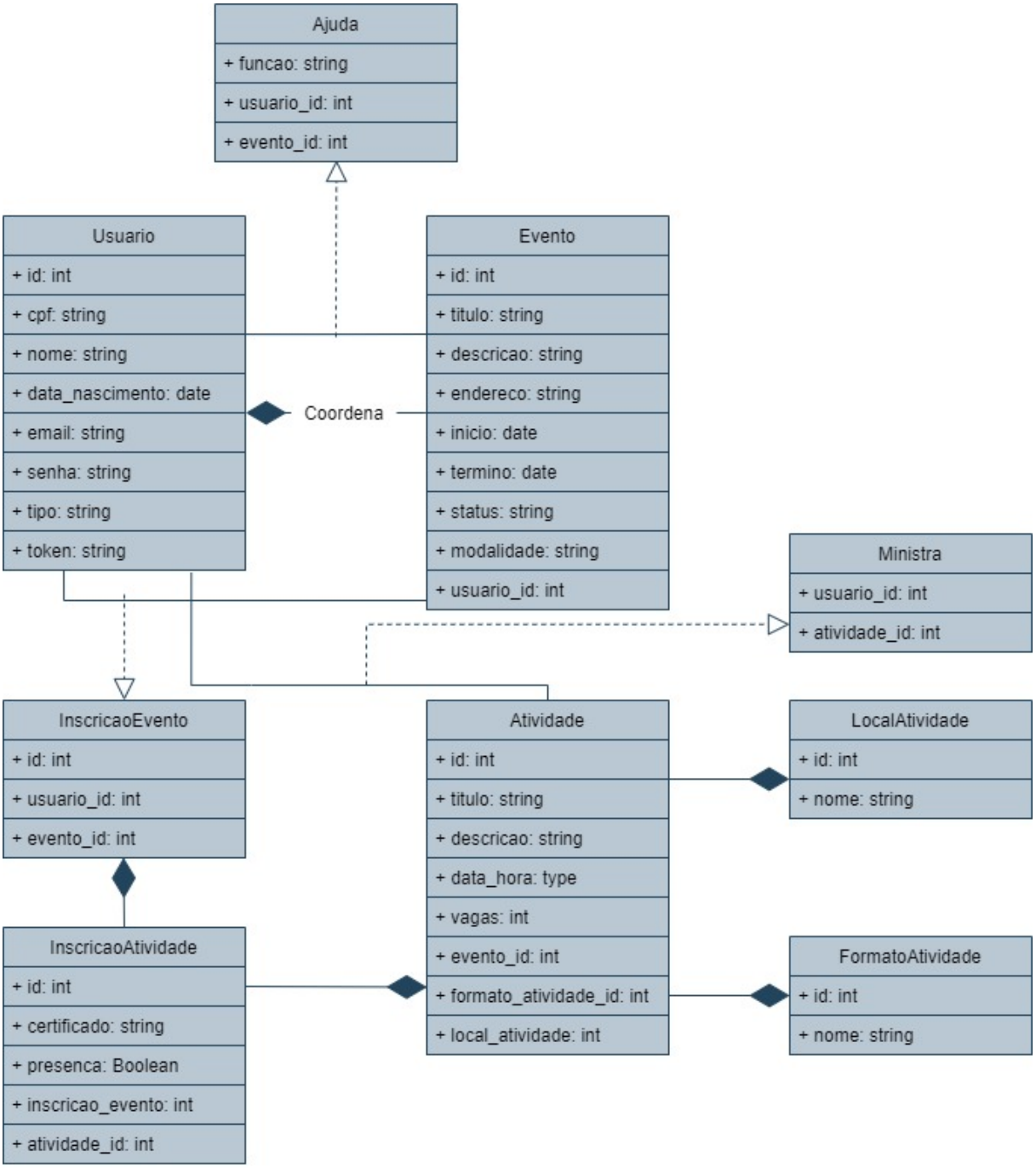

Figura 3 – Diagrama de classes

Outro diagrama desenvolvido foi o de atividade, responsável por demonstrar o fluxo de execução do sistema. A figura 4 demonstra o diagrama de atividade da criação de eventos.

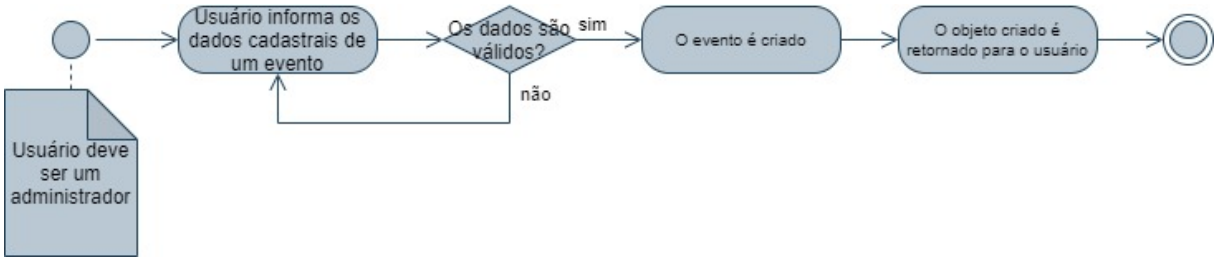

Figura 4 – Diagrama de atividade - criação de evento

Além destes, modelou-se também o diagrama de sequência, conforme apresentado na figura 5, o qual descreve a sequência de processos na criação de um usuário.

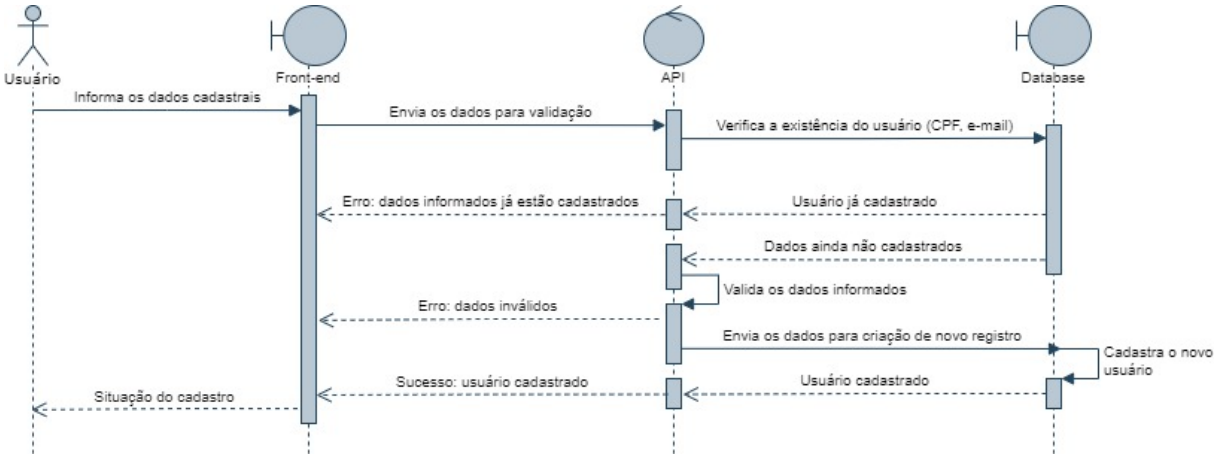

Figura 5 – Diagrama de sequência - cadastro de usuário

3.3 Requisitos

O levantamento de requisitos foi realizado juntamente ao *product-owner* do projeto. Os requisitos foram definidos com base no protótipo da aplicação[9]. Esse momento permitiu também a coleta de informações sobre a estrutura dos dados. As tabelas 1 e 2 apresentam os requisitos funcionais e não funcionais, respectivamente.

[9] Disponível em https://www.figma.com/file/dl5r1NvDF9RChVsjqZe5O2/Gerenciador-de-Eventos-2021

RF	Descrição
	O sistema deve permitir a realização de inscrição em eventos e em atividades
	O sistema deve permitir a busca de eventos
	A busca de eventos pode ser filtrada por nome, status e modalidade
	Os filtros devem funcionar em conjunto
	As atividades devem ter um ministrante
	As atividades devem ter um tipo (palestra, oficina, etc.)
	Para se matricular em uma atividade deve-se, obrigatoriamente, estar matriculado em seu evento
	Usuários com o domínio do e-mail "@ifro.edu.br" são organizadores
	Apenas organizadores são capazes de criar eventos, logo, atividades
	O sistema deve gerar códigos de certificados para as atividades
	O sistema deve possuir uma ferramenta de validação de códigos de certificados

Tabela 1 – Requisitos funcionais

RQ	Descrição
	O sistema deve ser desenvolvido na linguagem de programação JavaScript

Tabela 2 – Requisitos não funcionais

Vale lembrar que os requisitos aqui destacados referem-se a apenas uma parte da aplicação. Alguns outros foram levantados, entretanto, não eram abrangidos pelo escopo deste projeto.

3.4 Arquitetura do software

A arquitetura dessa aplicação foi baseada na Arquitetura Cliente/Servidor. Utilizando essa arquitetura, visa-se estruturar o sistema como um conjunto de processos que cooperarão para entregar serviços a um usuário. Nesse modelo, geralmente utiliza-se protocolos de comunicação do tipo requisição/resposta . Com o objetivo de receber algum serviço, o cliente envia uma requisição ao servidor. Por sua vez, o servidor processa as informações associadas ao serviço e retorna uma resposta ao cliente.

A API desenvolvida servirá de conexão entre o *front-end* e o banco de dados. A figura 6 demonstra a arquitetura do sistema.

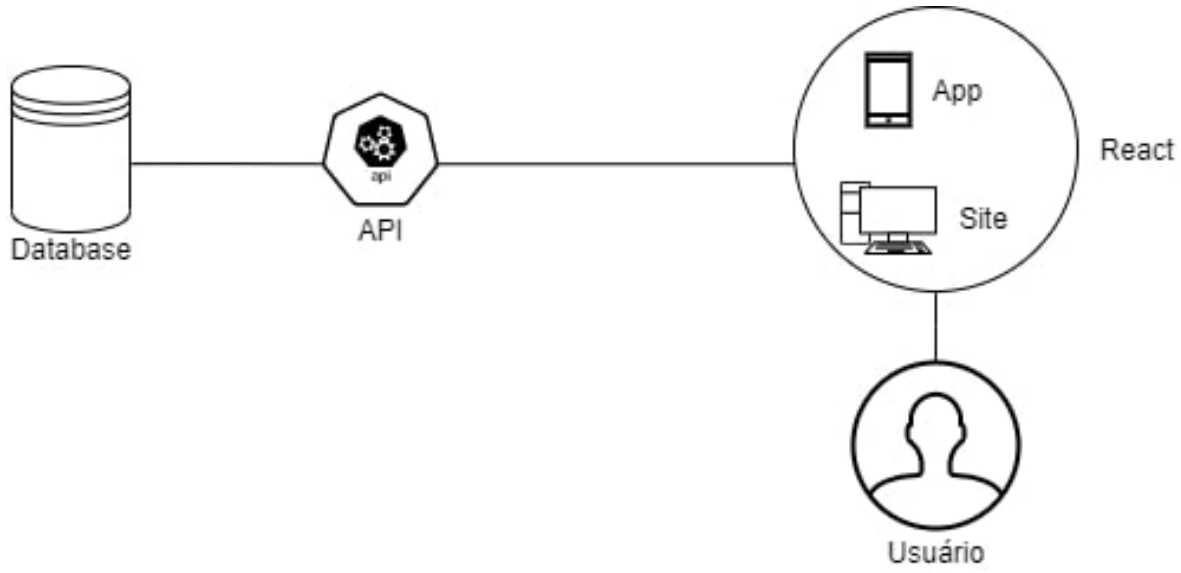

Figura 6 – Arquitetura

3.5 Licença de uso

A licença utilizada no desenvolvimento desse *software* foi a licença "MIT"[10]. Uma licença para *software* livre desenvolvida pelo Instituto de Tecnologia de Massachusetts.

[10] Disponível em https://opensource.org/licenses/MIT

4 Resultados e discussões

Neste capítulo são apresentados os resultados e discussões acerca da aplicação proposta.

4.1 Gerenciamento de tarefas - Métricas

4.1.1 Lead Time e Cycle Time

O *lead time* refere-se a todo o período desde a criação da tarefa até a sua conclusão e entrega. Já o *cycle time* diz respeito ao tempo em que a tarefa ficou em desenvolvimento.

As figuras 7, 8, 9 mostram os *lead times* referentes aos meses de setembro, outubro e novembro, respectivamente.

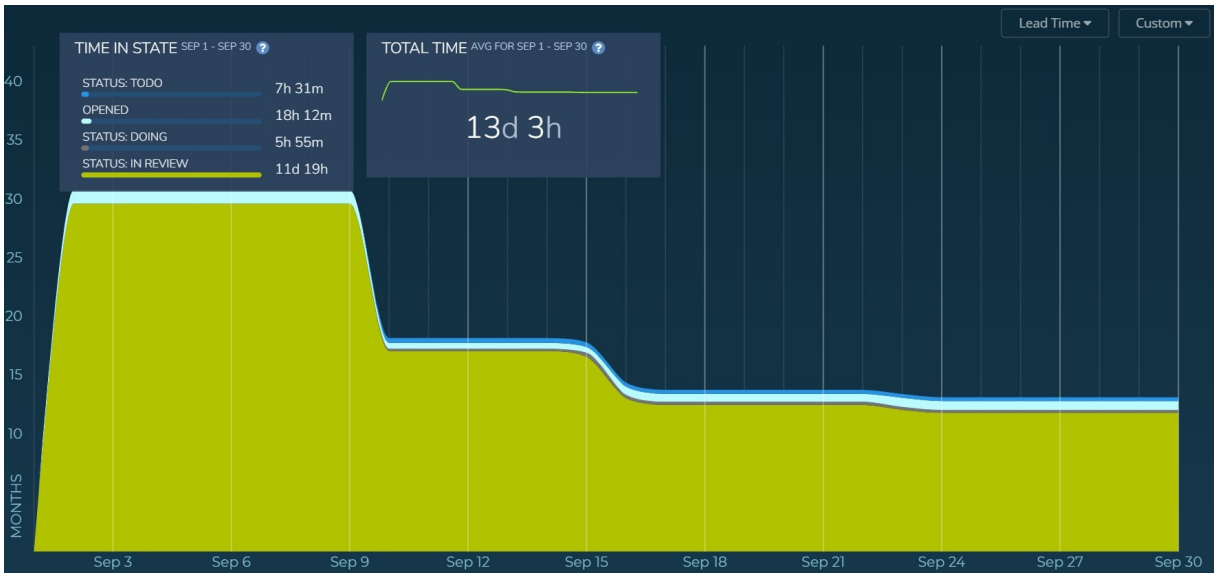

Figura 7 – *Lead Time* - Setembro

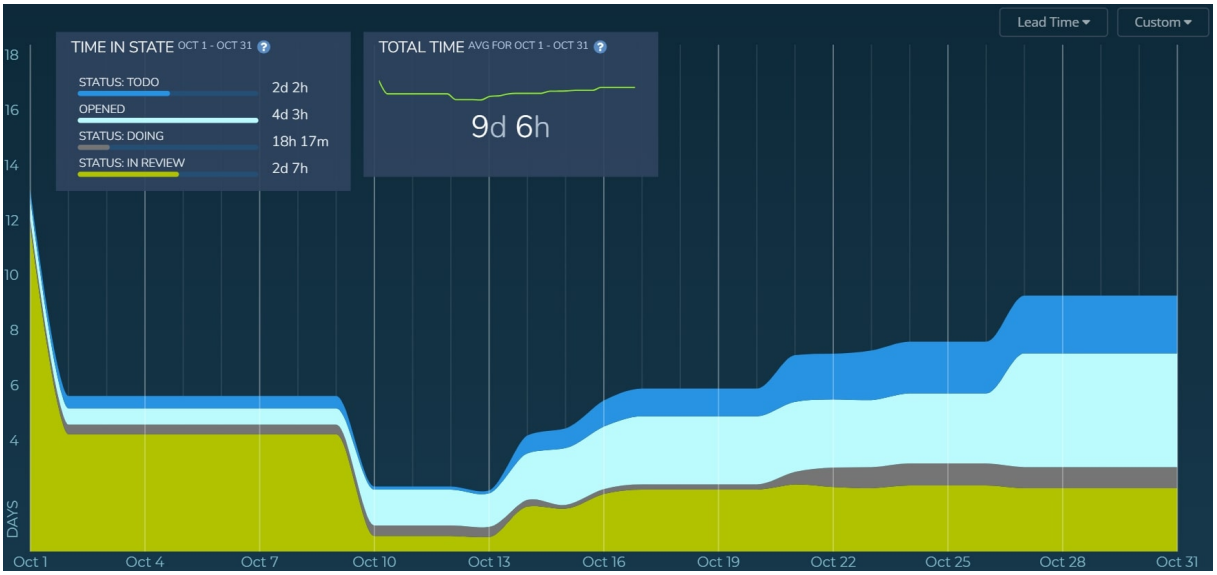

Figura 8 – *Lead Time* - Outubro

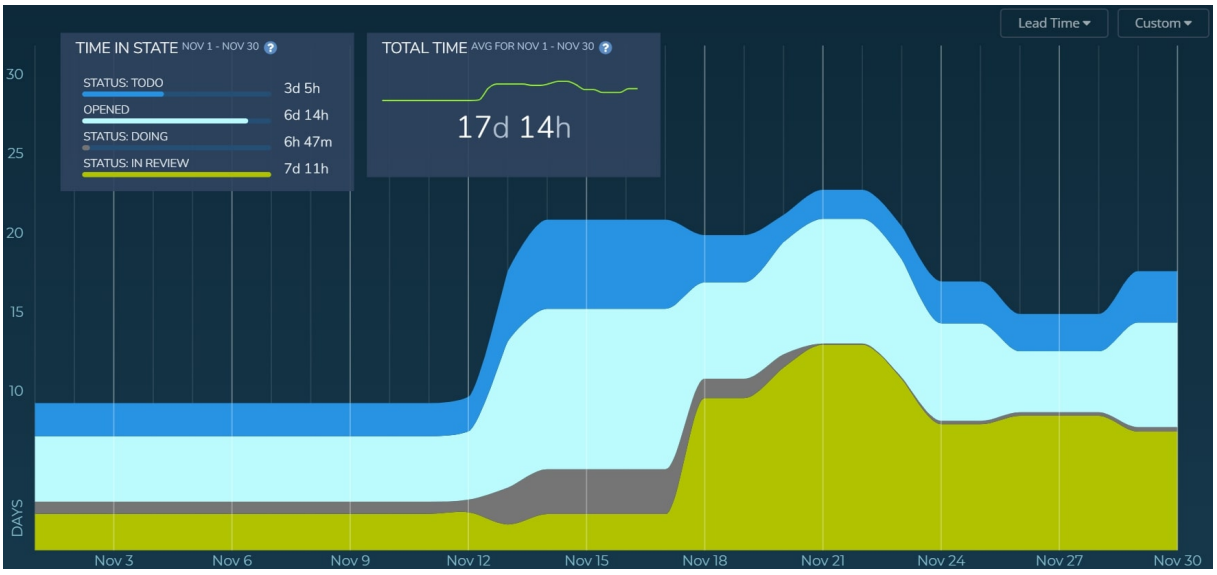

Figura 9 – *Lead Time* - Novembro

As figuras 10, 11, 12 mostram os *cycle times* referentes aos meses de setembro, outubro e novembro, respectivamente.

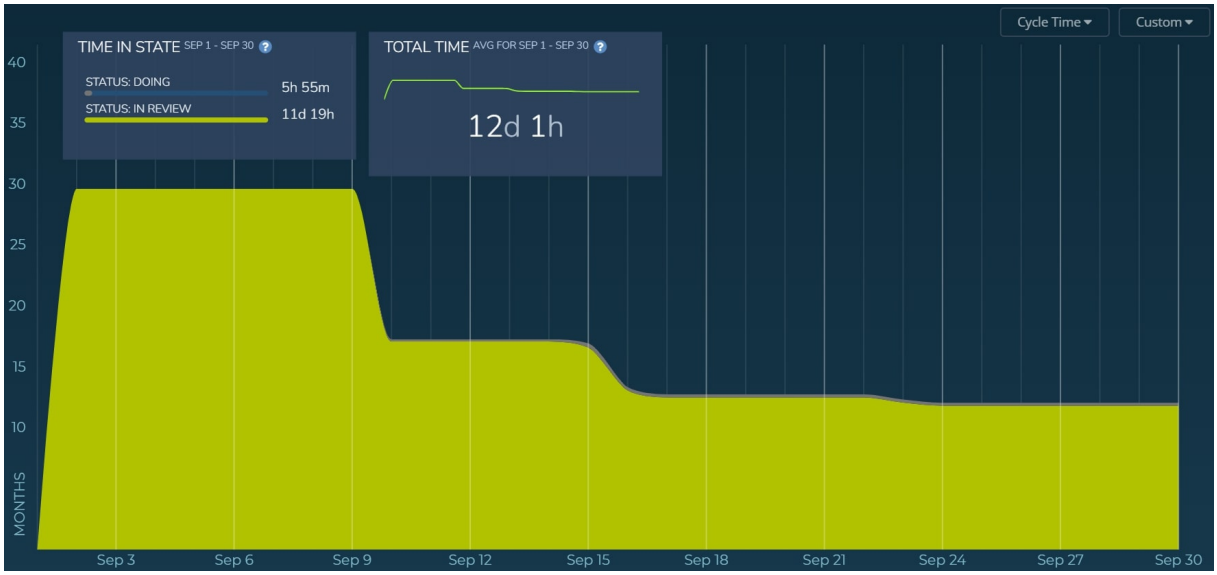

Figura 10 – *Cycle Time* - Setembro

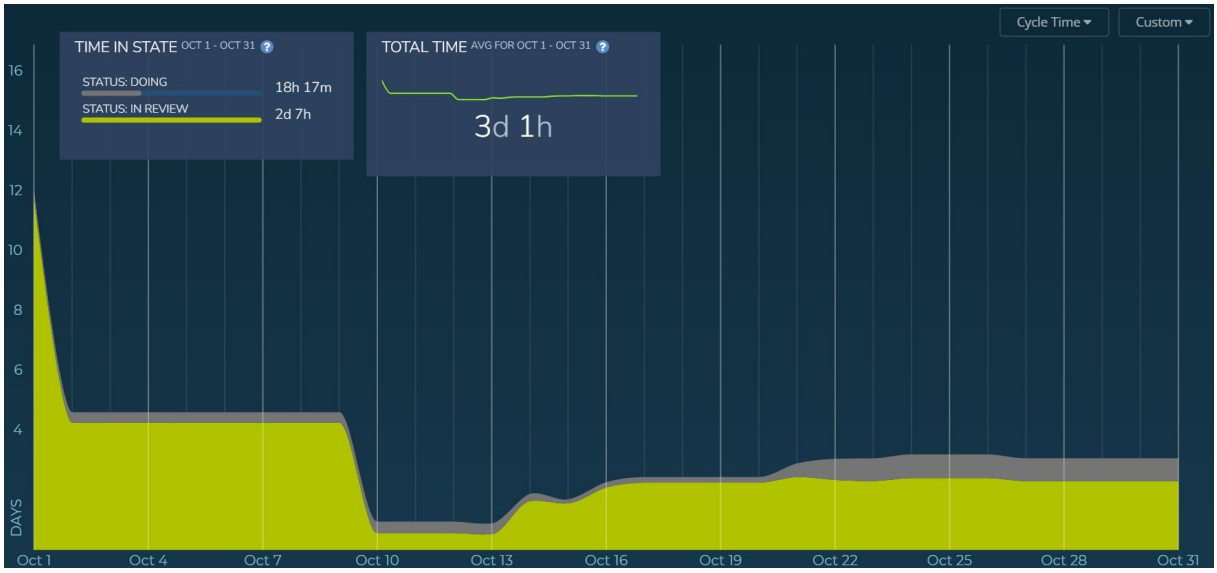

Figura 11 – *Cycle Time* - Outubro

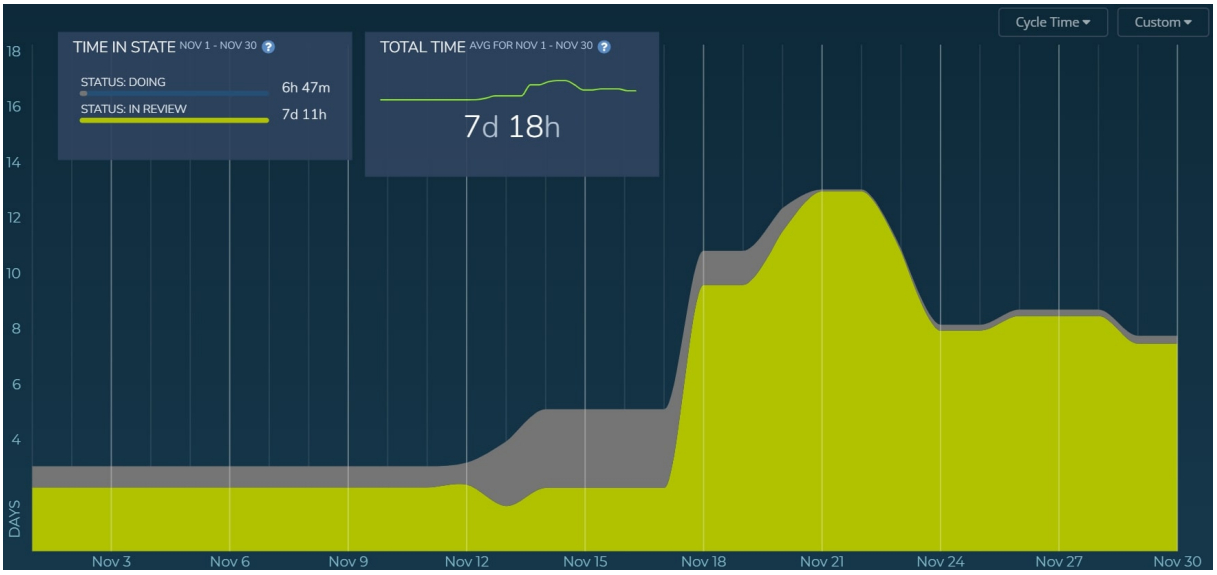

Figura 12 – *Cycle Time* - Novembro

A partir dos gráficos nota-se que o mês de setembro teve um começo com um alto *lead time* e *cycle time*, isso ocorreu pois no começo do processo de desenvolvimento, haviam muitas tecnologias novas para serem assimiladas. Outro fator que contribuiu para esse número foi o fato de que nesse momento, as atividades, para que pudessem ser consideradas concluídas, precisavam ser validadas juntamente ao *product-owner*. Essa validação ocorria uma vez a cada duas semanas, ocasionando em um aumento no valor de ambas as métricas. Após a primeira semana de setembro, houve uma redução significativa das mesmas, o que se repetiu na terceira semana, entretanto com menos intensidade.

No mês de outubro, nota-se uma diminuição nas métricas. Nesse mês, quase todas as atividades desenvolvidas foram relacionadas a funcionalidades implementadas na aplicação, portanto, todo o processo fluiu mais tranquilamente, acarretando em um gráfico bem mais estável quando comparado ao do mês anterior.

No início do mês de novembro - último mês de desenvolvimento da aplicação, o processo manteve os valores do *lead time* e *cycle time* constantes. A partir da metade do mês, como o processo de desenvolvimento estava chegando ao fim, resolvi dedicar mais tempo à produção da monografia, acarretando em atividades que demoravam mais para serem iniciadas.

A partir das figuras 10, 11, 12, nota-se que o tempo que a atividade esperava para ser testada, ou validada, era muito superior ao tempo em que a mesma ficava em desenvolvimento.

4.1.2 Taxa de transferência

A taxa de transferência é uma relação entre as atividades desenvolvidas e uma unidade de tempo - no caso deste trabalho, um mês. A figura 13 expõe a taxa de transferência durante o processo de desenvolvimento

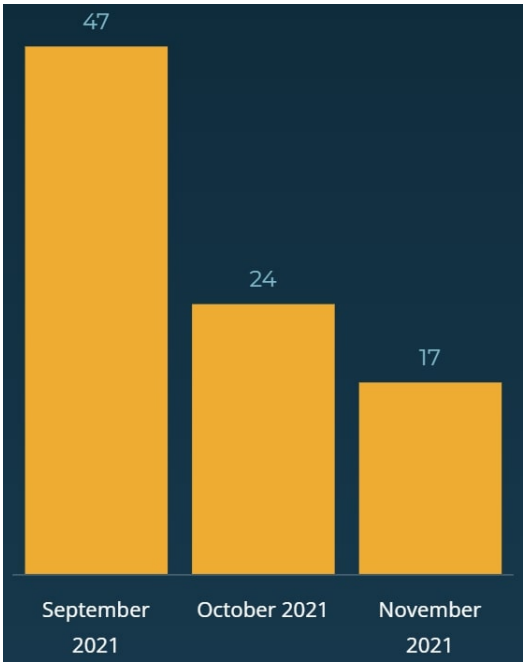

Figura 13 – *Throughput*

A partir da figura 13, é possível notar que o mês com maior volume de desenvolvimento foi setembro, isso porque, nesse mês, muitas atividades mais fáceis e menores foram desenvolvidas, como a preparação do ambiente de desenvolvimento e a criação de diagramas. Após esse momento inicial, as funcionalidades mais complexas da aplicação começaram a ser desenvolvidas, acarretando em um maior tempo necessário para a conclusão de uma atividade.

4.2 Relatório de testes

Durante o desenvolvimento da aplicação utilizamos duas abordagens de testes: testes automatizados e testes manuais. Para os testes manuais, utilizou-se a extensão Thunder Client[1]. Para os testes automatizados, os *frameworks* Jest[2] e Supertest[3] foram utilizados. Além dessas ferramentas, a base de dados citada anteriormente foi de fundamental importância para a execução dessa etapa, já que todas as rotas utilizam conexão com a mesma para persistir os dados.

4.2.1 Testes manuais

Para testar as funcionalidades desenvolvidas, primeiramente realizava-se o teste manual. A cada semana, toda a equipe de desenvolvimento realizava testes para verificar o funcionamento das funcionalidades desenvolvidas. Os testes validavam situações de sucesso ou de erro. Para

[1] Disponível em https://www.thunderclient.io
[2] Disponível em https://jestjs.io/pt-BR/
[3] Disponível em https://www.npmjs.com/package/supertest

facilitar a realização desse método, utilizou-se a extensão Thunder Client, para Visual Studio Code. Essa extensão consiste em um *Rest Client* voltado para teste de APIs.

O principal motivo pela escolha dessa ferramenta foi a praticidade. Por ser uma extensão, ela funcionava na mesma *interface* que o editor de código-fonte, não necessitando a utilização de um *software* a parte. Além disso, ela permite que exportemos um arquivo *JSON* contendo todas as requisições utilizadas para os testes. A partir desse arquivo, é possível importar todas as requisições desenvolvidas - processo que facilitou a execução de testes manuais.

4.2.2 Testes automatizados

Para os testes automatizados, arquivos reesposáveis pela declaração dos testes foram criados. Os testes foram desenvolvidos de acordo com a sintaxe padrão do *framework* Jest.

A sintaxe consiste em:

- Declaração da descrição com o método “describe”, onde informa-se a descrição dos testes a serem realizados, de uma forma geral. Esse método também invoca uma função de *callback*, a qual contém todos os testes que serão realizados;
- Declaração de testes com o método “it”, no qual também informa-se a descrição do comportamento esperado do teste a ser realizado, entretanto, dessa vez, mais especificamente. Assim como o “describe”, o método também invoca um função de *callback*, a qual contém a lógica do teste.

O *framework* Supertest é utilizado na criação de uma constante, a qual é responsável por realizar uma requisição a uma rota específica e armazenar a resposta da mesma. Após a resposta, o valor é utilizado para a lógica do teste.

A figura 14 demonstra a sintaxe e lógica de um dos testes desenvolvidos.

api-plataforma-eventos-ifro auth.test.js

```
require('dotenv').config()
const app = require('../src/index')

let request = require('supertest')
request = request(app)


describe('Testando a funcionalidade de autenticação', () => {
  it('A função de login deve retornar um status 200', async () => {
    const response = await request.post('/api/v1/login').send({
      email: 'wandreus@ifro.edu.br',
      senha: 'teste123'
    })
    expect(response.statusCode).toBe(200)
  })

  it('A função de login deve retornar um status 400', async () => {
    const response = await request.post('/api/v1/login').send({
      email: 'wandreus@ifro.edu.br',
      senha: 'teste1234'
    })
    expect(response.statusCode).toBe(400)
  })
})
```

Figura 14 – Teste automatizado - *Login*

Na tabela 3 estão expostos os testes desenvolvidos, bem como os *status* HTTP esperados em caso de sucesso ou erro.

Rota	Status - Sucesso	Status - Erro
POST /login - Funcionalidade de login	200	400
POST /cadastro - Função de cadastro de usuários	201	400
GET /usuario - Listagem dos dados do usuário logado	200	500, 403
GET /evento - Listagem de eventos	200	404, 403

Tabela 3 – Testes e retornos esperados

4.3 Documentação

Visando melhorar a experiência de uso de futuros desenvolvedores que trabalharão nesse projeto, foi desenvolvida a documentação para essa API. Nessa etapa, utilizou-se do *software open-source* Swagger[4]. Com o auxílio da biblioteca Swagger UI Express[5], é possível fornecer uma documentação gerada por Swagger UI, baseada em um arquivo que pode ser YAML ou JSON. Isso resulta em uma documentação da API hospedada no servidor da aplicação por meio de uma rota[6].

Conforme mostrado na figura 15, as rotas documentadas são separadas por *tags* definidas no código da documentação, nesse caso, as rotas estão divididas em:

- Atividade: categoria que contém todas as rotas relacionadas com atividades;
- Certificado: categoria que contém todas as rotas relacionadas com certificados;
- Evento: categoria que contém todas as rotas relacionadas com eventos;
- Usuário: categoria que contém todas as rotas relacionadas com usuários.

Figura 15 – Tags Swagger

Além dessa separação, cada rota contém um *summary* (resumo), o qual também é definido no código da documentação. Esse resumo serve para descrever a funcionalidade de determinada rota.

As rotas também possuem o tipo de requisição, nessa aplicação foram utilizados apenas quatro: Delete; Get; Post; Put.

- Delete: rotas que excluem algum registro, nessas rotas informa-se na URL o identificador do registro a ser excluído. A figura 16 demonstra um exemplo dessas rotas;

[4] Disponível em https://swagger.io
[5] Disponível em https://www.npmjs.com/package/swagger-ui-express
[6] Disponível em https://eventos.fslab.dev/docs

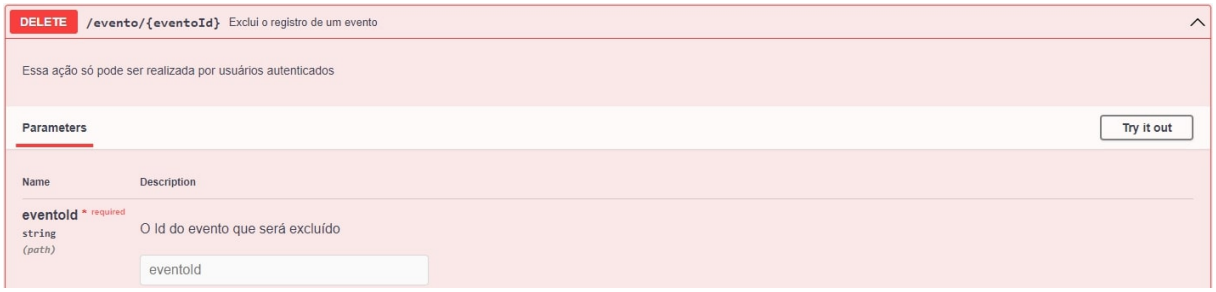

Figura 16 – Rota DELETE

- Get: rotas que retornam informações ao *front-end* (um registro, por exemplo). A figura 17 demonstra um exemplo dessas rotas;

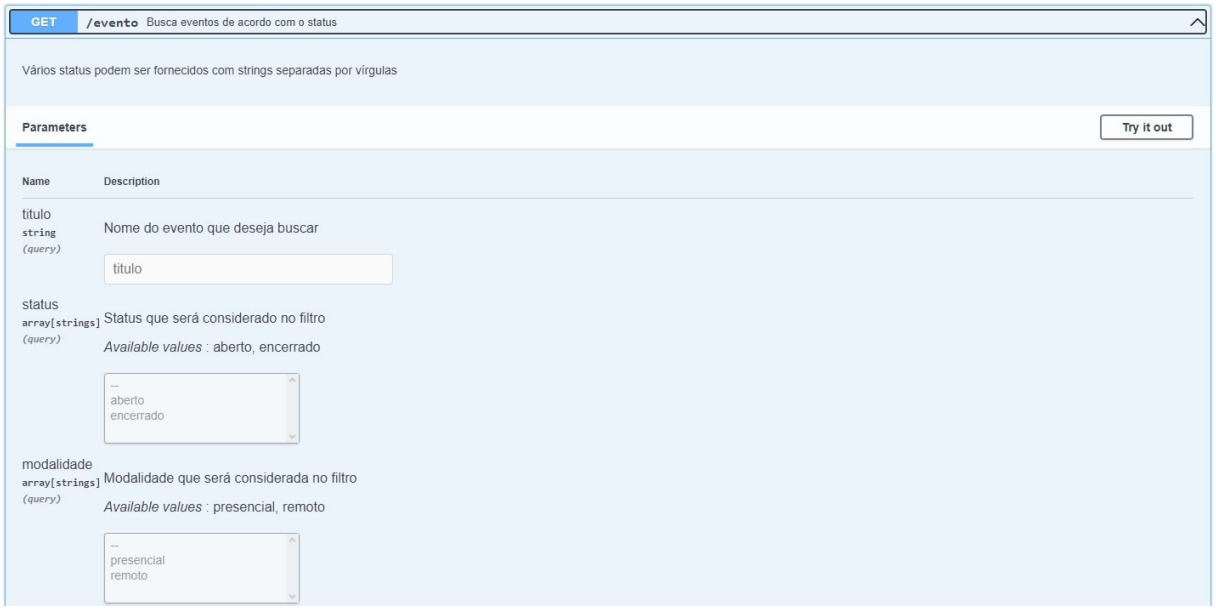

Figura 17 – Rota GET

- Post: rotas que enviam informações do *front-end* ao *back-end*, foram utilizadas, principalmente, para a criação de registros. Nessas rotas as informações são passadas no *body* (corpo) da requisição. A figura 18 demonstra um exemplo dessas rotas;

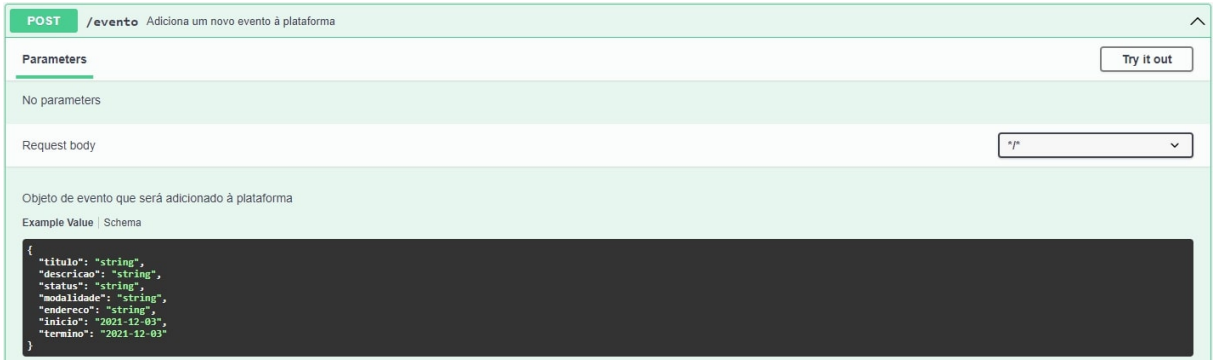

Figura 18 – Rota POST

- Put: rotas que atualizam registros já existentes. Nessas rotas informa-se no corpo da requisição as informações à serem atualizadas. Além disso, na URL é informado o identificador do registro a ser alterado. A figura 19 demonstra um exemplo dessas rotas.

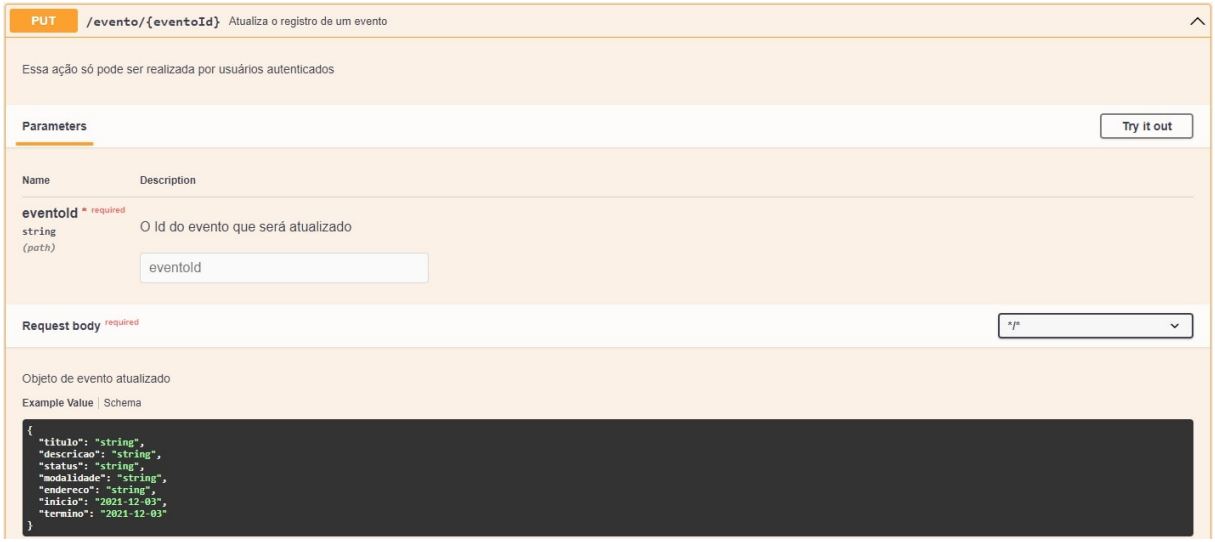

Figura 19 – Rota PUT

Há casos de rotas que possuem mais de um tipo de requisição, ocasionando em rotas "diferentes" com o mesmo URL, como demonstrado na figura 20

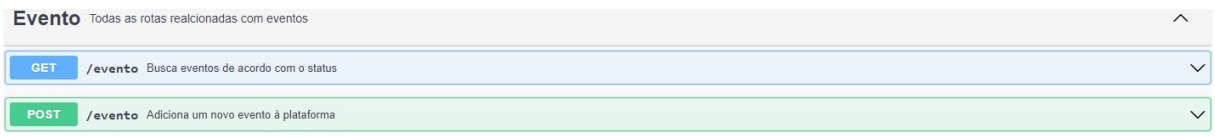

Figura 20 – Rota "*/evento*"

4.4 Populando o banco

Para facilitar a verificação do funcionamento das rotas desenvolvidas - principalmente as que retornam registros - foram desenvolvidos arquivos para semear o banco de dados. Com o auxílio da biblioteca Faker[7], tornou-se possível gerar registros com informações falsas, os quais eram persistidos no banco de dados e utilizados para verificar o retorno das rotas. Ao executar o comando correto, a aplicação inicia o processo, gerando registros para todas as tabelas, exceto para as que armazenam relacionamentos. A figura 21 demonstra o arquivo de geração de usuários.

api-plataforma-de-eventos-ifro - usuarios.js

```
const faker = require('faker')
faker.locale = 'pt_BR'
const gerarCPF = require('gerar-cpf')
const models = require('../src/database/models.js')

let n = 1
while (n <= 100) {
  try {
    models.Usuario.create({
      nome: faker.name.findName(),
      cpf: gerarCPF(),
      email: faker.internet.email(),
      senha: faker.internet.password(),
      foto_perfil: faker.internet.avatar(),
      data_nascimento: faker.date.between('1935-01-01', '2007-01-01'),
      tipo: faker.random.arrayElement(['padrão', 'admin']),
      createdAt: new Date(),
      updatedAt: new Date()
    })
  } catch (error) {
    console.log({
      message: error.message
    })
  }
  n++
}
```

Figura 21 – Seeder - Usuários

[7] Disponível em: https://www.npmjs.com/package/faker

4.4.1 Validação de entrada

A fim de evitar o cadastro de informações inválidas, o projeto conta com validadores de entrada. Os validadores se encontram em arquivos separados, sendo que, para cada tabela em que o usuário pode realizar a inserção ou atualização de dados, existe um validador. Cada qual constitui-se de uma única função que analisa os dados que podem ser enviados pelo usuário (não é necessário validar um id, já que o usuário não consegue controlá-lo). Essa função recebe dois parâmetros: o objeto a ser validado e o método de validação. Caso o método de validação seja "create", a função valida todos os dados, retornando qualquer erro; Caso o método seja "update", a função valida apenas os dados informados pelo usuário, limpando os dados que não podem ser atualizados.

A figura 22 mostra um exemplo da validação realizada pelo validador de atividades.

api-plataforma-eventos-ifro - validatorAtividade

```
// Validação da data
    if (!atividade.data_hora && action !== 'update') {
      // Verifica se a data e o harário foram informados
      errors.push('
Para cadastrar uma atividade, é necessário informar, ao menos, uma data e horário
 (início e fim)
')
    } else if (!atividade.data_hora && action === 'update') {
      // Não é necessária nenhuma ação
    } else {
      atividade.data_hora.forEach(data => {
        // Separação da data e hora
        let horarioInicio = data.inicio.split(' ')
        let horarioTermino = data.termino.split(' ')

        // Atribui as datas a variáveis
        let dataInicio = horarioInicio[0]
        let dataTermino = horarioTermino[0]

        // Convertendo as datas para o formato do banco de dados
        dataInicio = dataInicio.split('/')
        dataTermino = dataTermino.split('/')
        data.inicio = dataInicio[2] + '-' + dataInicio[1] + '-' + dataInicio[0] + '
T' + horarioInicio[1]
        data.termino = dataTermino[2] + '-' + dataTermino[1] + '-' + dataTermino[0
] + 'T' + horarioTermino[1]

        // Cria objetos de data com as datas informadas
        horarioInicio = new Date(data.inicio)
        horarioTermino = new Date(data.termino)

        // Valida as datas informadas
        if (horarioInicio.getTime() === isNaN) {
          errors.push('Formato de data de início (no ' + (atividade.data_hora.
indexOf(data) + 1) + 'º objeto) inválido. Deve ser (dd/mm/aaaa HH:MM)')
        }
        if (horarioTermino.getTime() === isNaN) {
          errors.push('Formato de data de término (no ' + (atividade.data_hora.
indexOf(data) + 1) + 'º objeto) inválido. Deve ser (dd/mm/aaaa HH:MM)')
        }

        // Verifica se a data de início é anterior a data de término
        if (horarioInicio > horarioTermino) {
          errors.push('
A data de início da atividade não pode ser posterior à data de término (no ' + (
atividade.data_hora.indexOf(data) + 1) + 'º objeto)')
        }
```

Figura 22 – Validação de atividades

Após verificar a validade de um dado, a função adiciona uma mensagem de erro à um vetor *errors*, caso o dado seja inválido. Ao final da execução da função, verifica-se o tamanho deste vetor, caso seja diferente de 0 (zero), a função retorna-o, indicando todos os erros de entrada para o usuário; caso o vetor esteja vazio, a função retorna *true*, permitindo que o cadastro seja criado, ou alterado.

4.5 Implantação

Para o *deploy* da aplicação utilizou-se a mesma técnica empregada na hospedagem do SGBD - a conteinerização. Todos os serviços da aplicação estão sendo executados em uma VM, a qual está rodando no servidor da FSLab[8]. Essa técnica foi escolhida tendo em vista a facilidade de escalonamento da aplicação, além da redução de recursos consumidos, em razão da utilização de contêineres.

Para a execução da aplicação e gerenciamento de seus processos, utilizou-se a ferramenta PM2[9]. A escolha dessa ferramenta deu-se em decorrência de sua gama de funcionalidades. Além de geração de relatórios e métricas, o PM2 fornece reinício automático da aplicação. Outro poderoso recurso dessa ferramenta é o *Cluster Mode*, permitindo o dimensionamento dinâmico da aplicação para cada CPU disponível.

4.6 Demonstração do *software*

As funcionalidades da API podem ser testadas através de uma plataforma para a realização de requisições HTTP, durante o desenvolvimento utilizou-se a extensão Thunder Client, como citado anteriormente.

Para testar as rotas, deve-se criar um usuário através da rota POST "*/cadastro*", é recomendado que o e-mail informado durante a criação tenha o domínio "@ifro.edu.br", para que possa verificar todos os recursos. Após a criação desse usuário, deve-se logar na plataforma, informando o e-mail e senha do usuário criado. O *login* pode ser realizado através da rota "*/login*". É válido lembrar que a aplicação também realiza o *hash* da senha do usuário. Caso o *login* seja bem-sucedido, a resposta da rota possuirá um *token* de autenticação. Nas demais rotas, o *token* deve ser informado no cabeçalho da requisição no parâmetro "*x-api-token*".

Através da rota POST - "*/evento*", é possível criar um evento. Conforme demonstrado na figura 23, para realizar a criação de um evento é necessário informar seis campos: título; descrição; endereço; data de início; data de término e modalidade. Após o envio da requisição, os dados informados serão validados. Em caso de dados inválidos, os erros existentes serão retornados na resposta da rota, conforme evidenciado na figura 24.

[8] Disponível em https://eventos.fslab.dev
[9] Disponível em https://pm2.keymetrics.io

Figura 23 – Requisição - POST “/evento”

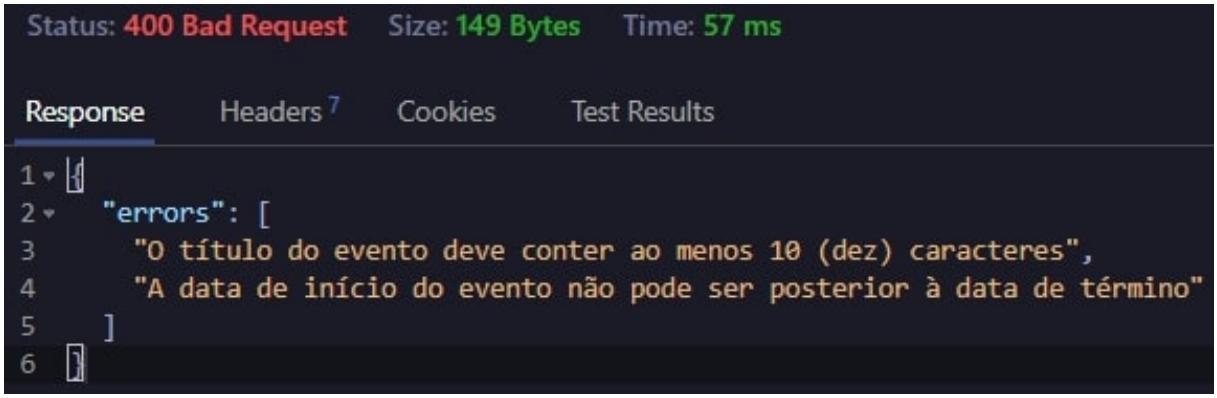

Figura 24 – Resposta - POST “/evento” - Erro

Após a criação de um evento, é possível visualizá-lo na resposta de uma requisição na rota GET - “*/evento*”, a qual retorna os eventos cadastrados na plataforma, conforme demonstrado na figura 25, com um sistema de paginação. Essa mesma requisição pode ser enviada com parâmetros de *query*, os quais filtrarão a busca de eventos. Os eventos podem ser filtrados por: título, status e modalidade. Ainda na *query* é possível informar a página que deseja-se visualizar. A utilização desses parâmetros ocorrem como mostrado na figura 26

Status: 200 OK Size: 3.26 KB Time: 31 ms

Response Headers 7 Cookies Test Results


```
{
  "count": 109,
  "rows": [
    {
      "id": 1,
      "titulo": "Evento 1 - local",
      "descricao": "Evento sobre ",
      "endereco": "Avenida alguma coisa - 3244",
      "inicio": "2021-12-12T00:00:00.000Z",
      "termino": "2021-12-13T00:00:00.000Z",
      "status": "aberto",
      "modalidade": "presencial",
      "usuario_id": null,
      "createdAt": "2021-11-23T14:49:21.000Z",
      "updatedAt": "2021-11-23T14:50:59.000Z",
      "usuarioId": null
    },
    {
      "id": 2,
      "titulo": "Direct Response Executive",
      "descricao": "Et magni quia quae voluptatem qui sint aut.",
      "endereco": "705 Daniel Alameda Apt. 420",
      "inicio": null,
      "termino": null,
      "status": "aberto",
      "modalidade": "remoto",
      "usuario_id": null,
      "createdAt": "2021-11-24T14:10:11.000Z",
      "updatedAt": "2021-11-24T14:10:11.000Z",
      "usuarioId": null
```


Figura 25 – Resposta - GET “/evento” - Sucesso

GET http://127.0.0.1:7070/api/v1/evento?titulo=direct&status=aberto&modalidade=presencial&pagina=1 Send

Figura 26 – *Query* - GET “/evento”

Para a criação de uma atividade o processo é o mesmo da criação de eventos - informa-se os dados no corpo da requisição, os quais serão validados, porém essa requisição é feita em outra rota: POST - "*/evento/:evento_id/atividade*", na qual o parâmetro "*:evento_id*" refere-se ao evento que fornecerá esta atividade. A listagem de atividades é um pouco diferente, já que só é possível visualizar as atividades de um determinado evento por vez, conforme exposto na figura 27. Também não há um mecanismo de busca e filtro para atividades. O evento que terá suas atividades listadas é informado por meio do parâmetro - "*evento_id*".

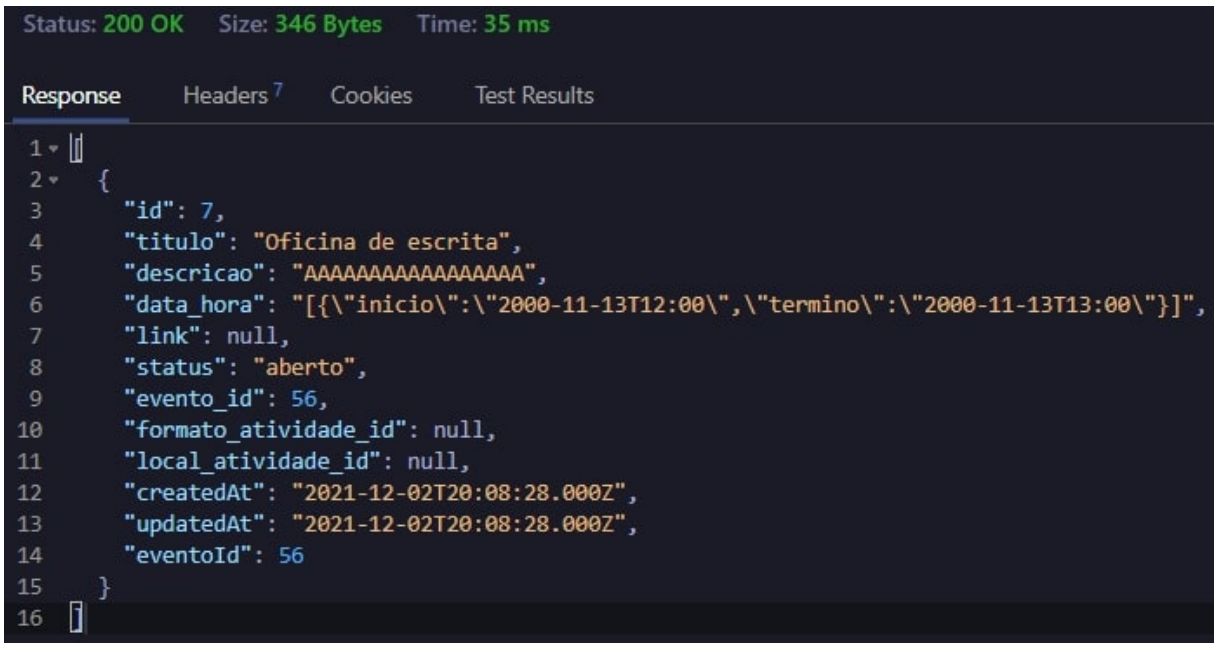

Figura 27 – Resposta - GET "/evento/:evento_id/atividade" - Sucesso

Com o evento criado, um usuário pode inscrever-se no mesmo, através da rota POST - "/evento/:evento_id/inscricao". Somente após isso, ele pode, da mesma forma, inscrever-se nas atividades deste mesmo evento. As inscrições ficam armazenadas em registros no banco de dados. No momento, a única utilidade da inscrição em um evento é a permissão de se inscrever nas atividades do mesmo. Já as inscrições em atividades são utilizadas para confirmar a presença de um usuário.

Conforme exemplificado na figura 28, no corpo da requisição é informado um *array* que possui os IDs das inscrições em eventos que compareceram na atividade. Cada valor desses será validado pela aplicação, verificando a existência, ou não, de uma inscrição. Se a inscrição existir, altera-se para *true* o campo "presenca" do registro correspondente a ela na tabela "*inscricao_atividade*". Essa requisição é feita na rota POST "*/evento/:evento_id/atividade/:atividade_id/inscritos*". Os inscritos em uma atividade podem ser vistos a partir da mesma rota, entretanto, o método de requisição será GET. A figura 28 também demonstra a resposta da rota.

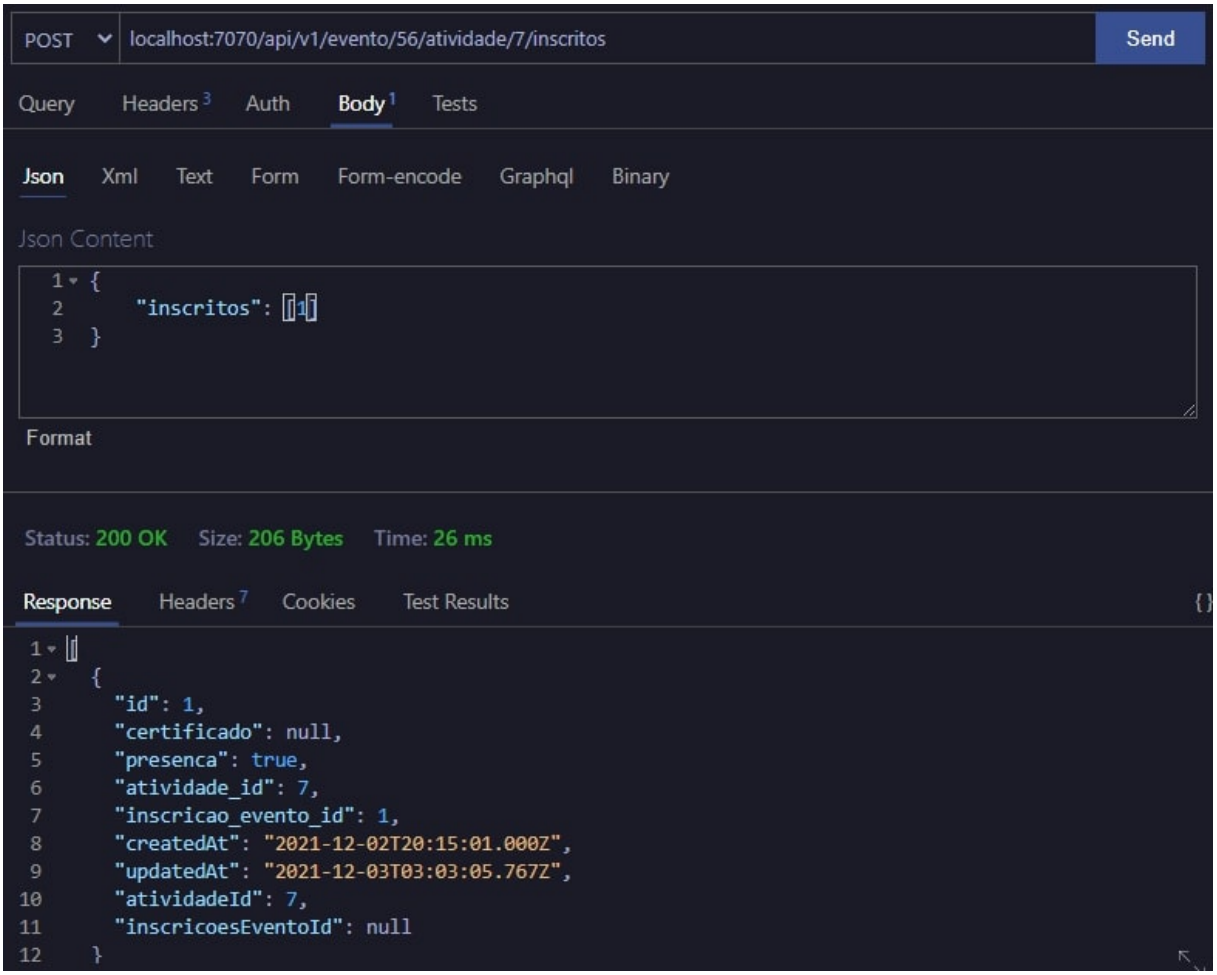

Figura 28 – Requisição - POST "/evento/:evento_id/atividade/:atividade_id/inscritos" - Sucesso

A partir da rota POST - "*/evento/:evento_id/atividade/:atividade_id*", todas as inscrições referentes a atividade definida pelo parâmetro "atividade_id" são trazidas do banco de dados. Após isso, através de uma estrutura de *loop*, verifica-se o valor do campo "presenca" de todos os registros. Os registros que tiverem a presença com valor *true* são atualizados, recebendo um UUID no campo "certificado". Esse UUID representa o código do certificado que será gerado por outra aplicação.

Os dados de um certificado - evento, atividade, usuário - podem ser verificados através da rota GET - "*/certificado/:certificado_id*", conforme mostrado na figura 29, nessa rota, por meio do parâmetro "certificado_id", informa-se o código do certificado que deseja verificar os dados. Os códigos de certificados também podem podem ser validados a partir da rota GET - "*/certificado/validar-certificado*", na qual o código do certificado a ser validado é informado no corpo da requisição. O código então é validado, caso haja uma inscrição com esse código, todos os dados referentes ao certificado são retornados na resposta da rota.

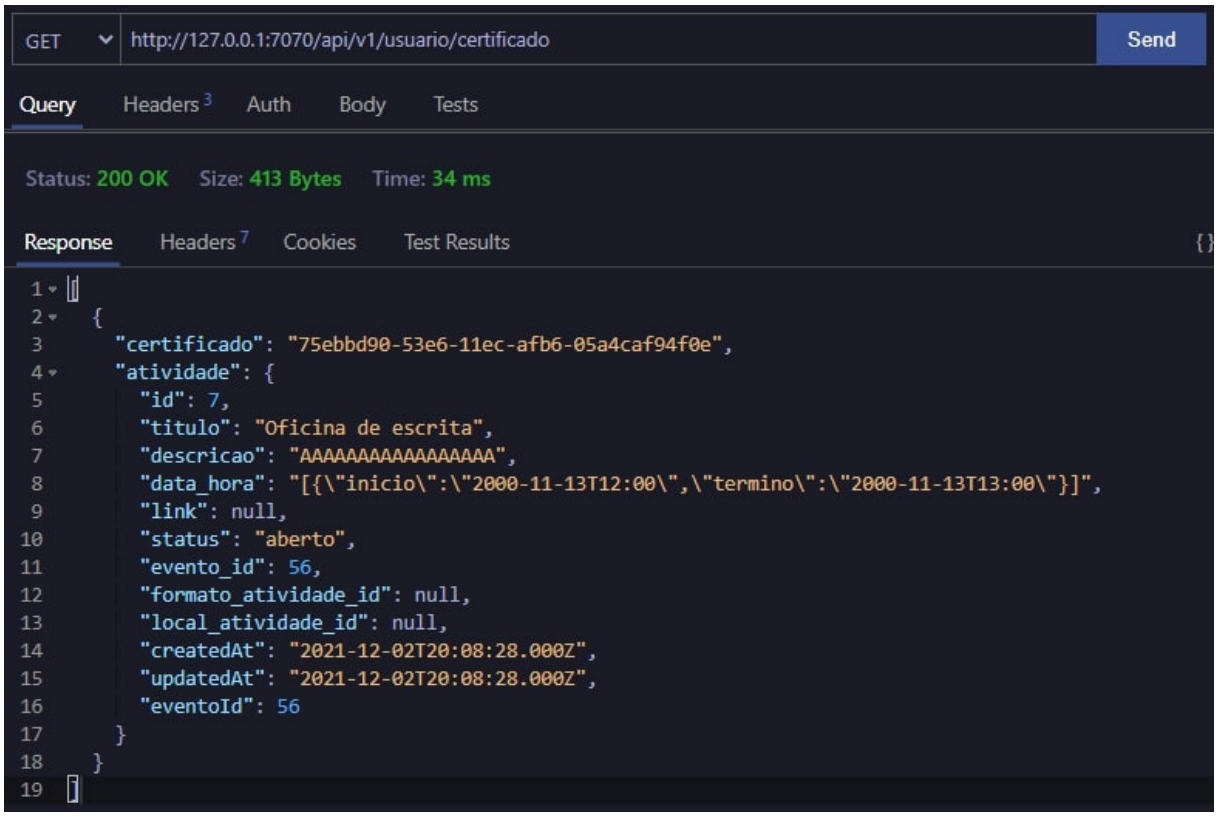

Figura 29 – Resposta - GET “/usuario/:usuario_id/certificado”

5 Considerações finais

O desenvolvimento desse projeto foi um grande desafio, graças a isso, um grande aprendizado. Durante todo o processo, tive de aprender novas tecnologias, boas práticas e metodologias utilizadas por profissionais. O produto, como dito anteriormente, é apenas uma parte da API da Plataforma de Eventos do IFRO, a qual não está completa. Entretanto, todo o escopo definido no início deste projeto foi atendido. Até o momento, muitas funcionalidades foram implementadas, as quais podem servir de base para o desenvolvimento futuro.

O projeto visa ajudar o IFRO durante o processo de gerenciamento de seus eventos institucionais. Ter uma plataforma própria, possibilitará o atendimento facilitado de seus requisitos. Além disso, por ser um *software open-source*, a plataforma pode ajudar instituições além do IFRO, um dos objetivos de um projeto de extensão.

5.1 Trabalhos futuros

Após o desenvolvimento da solução proposta, foram identificados os seguintes trabalhos futuros que podem ser realizados:

- Adicionar um usuário *admin*, o qual será responsável por:
  - Adicionar, atualizar e remover locais e formatos de atividades;
  - Verificar se os eventos estão de acordo com as regras da instituição para que os mesmos possam ser listados, já que, no momento, todos os eventos criados estão disponíveis na listagem.
- Permitir a implantação de novas instituições seria interessante, tendo em vista que a plataforma também disponibilizará eventos online
- Permitir a atribuição de ajudantes em um evento. Já foi implantada uma função bem parecida, na qual define-se os ministrantes de uma atividade
- Adicionar as funcionalidades referentes a trabalhos científicos.
- Adicionar a funcionalidade de enceramento de eventos.

Referências

ANDERSON, D. J. *Kanban: Mudança Evolucionária de Sucesso para Seu Negócio de Tecnologia*. [S.l.]: Blue Hole Press, 2011. v. 6. Citado na página 27.

BECK, K. et al. *Manifesto para Desenvolvimento Ágil de Software*. 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Citado na página 26.

CHACON, S.; STRAUB, B. *Pro Git*. [S.l.]: Apress, 2014. v. 2. Citado na página 26.

DOCKER. *What is a Container? | App Containerization | Docker*. 2018. Disponível em: <https://www.docker.com/resources/what-container>. Citado na página 24.

DOITY. *Doity*. 2021. Disponível em: <https://doity.com.br>. Citado na página 28.

EVEN3. *Even3*. 2021. Disponível em: <https://www.even3.com.br>. Citado na página 27.

EVENTIZE! *eventize!* 2021. Disponível em: <https://www.eventize.com.br>. Citado na página 28.

FLANAGAN, D. *JavaScript: O Guia Definitivo*. [S.l.]: BOOKMAN EDITORA LTDA, 2013. v. 6. Citado 2 vezes nas páginas 23 e 24.

IBM. *O que é o Docker? - Brasil | IBM*. 2021. Disponível em: <https://www.ibm.com/br-pt/cloud/learn/docker>. Citado na página 25.

KANBANIZE. *O que é Kanban? Definição e Detalhes Explicados | Kanbanize*. 2021. Disponível em: <https://kanbanize.com/pt/recursos-kanban/primeiros-passos/o-que-e-kanban>. Citado na página 27.

LACERDA, E. C. *Sistemas de Controle de Versão: Gerenciando versões de um documento*. 2012. Disponível em: <https://www.devmedia.com.br/sistemas-de-controle-de-versao/24574>. Citado na página 25.

MARIADB. *About MariaDB Server - MariaDB.org*. 2021. Disponível em: <https://mariadb.org/about>. Citado na página 25.

MOZILLA. *API - Glossário*. 2021. [Online; acessado em 27-Novembro-2021]. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Glossary/API>. Citado na página 23.

MOZILLA. *JavaScript*. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Citado na página 23.

NODE.JS. *Sobre | Node.js*. 2021. Disponível em: <https://nodejs.org/pt-br/about/>. Citado na página 24.

RED HAT. *O que é API?* 2017. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Citado na página 23.

SYMPLA. *Sympla*. 2021. Disponível em: <https://www.sympla.com.br>. Citado na página 27.

6 Licença MIT