PEDRO HENRIQUE DE MENEZES

RELATÓRIO DE ESTÁGIO: DESENVOLVIMENTO BACK-END NA EMPRESA IOASYS

LAVRAS – MG

2023

PEDRO HENRIQUE DE MENEZES

RELATÓRIO DE ESTÁGIO: DESENVOLVIMENTO BACK-END NA EMPRESA IOASYS

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. Antônio Maria Pereira de Resende
Orientador

LAVRAS – MG

2023

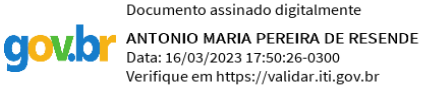
PEDRO HENRIQUE DE MENEZES

RELATÓRIO DE ESTÁGIO: DESENVOLVIMENTO BACK-END NA EMPRESA IOASYS

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 2 de Março de 2023.

Prof. Dr. Antônio Maria Pereira de Resende	UFLA
Prof. Dr. Bruno de Abreu Silva	UFLA
Prof. Dra. Paula Christina Figueira Cardoso	UFLA



Prof. Dr. Antônio Maria Pereira de Resende
Orientador

LAVRAS – MG
2023

Dedico este trabalho aos meus pais, por me proporcionarem esta oportunidade e priorizarem a minha formação acadêmica.

AGRADECIMENTOS

Agradeço, primeiramente a Deus, por me conduzir para as decisões corretas e por ser a base do meu conhecimento.

Aos meus pais, pela paciência, pelo incentivo e por serem a razão deste trabalho existir.

Ao meu irmão, pelos ensinamentos e por ser fonte de inspiração na vida profissional.

Aos meus amigos, João Vitor Rezende, Lucas Silvério e Thomaz Flanklin, pelo apoio técnico e pessoal durante o período do curso.

Ao professor Doutor Antônio Maria Pereira de Rezende, por aceitar me orientar durante este trabalho.

À empresa ioasys, por me conceder a oportunidade da primeira experiência profissional, pelo conhecimento disponibilizado, pelo tratamento fornecido aos colaboradores e por proporcionar um grande crescimento na minha carreira. Aos meus mentores técnicos, por me proporcionar conhecimento necessário para uma carreira profissional.

Enfim, agradeço a todos que contribuíram de alguma forma nesta fase decisiva da minha vida.

RESUMO

Este relatório apresenta a experiência em desenvolvimento back-end na empresa ioasys, desde o processo para ingressar na organização, como estagiário, até a finalização de um ano de trabalhos na implementação de projetos para diferentes tipos de clientes. O relato pretende esclarecer como a aplicação de conhecimentos adquiridos no setor acadêmico pode ser utilizado na prática e também apresentar as ferramentas de desenvolvimento utilizadas, como o trabalho com banco de dados relacionais e ORMs, versionamento Git, ferramentas AWS, a utilização de metodologias ágeis para a construção e desenvolvimento de projetos e a principal ferramenta nesse processo, o Node.js. Ao decorrer do processo do estágio, foi necessário enfrentar diversos desafios, entre esses, participar de um processo extenso para conquistar a vaga na organização e a adaptação dentro de um novo ambiente profissional.

Palavras-chave: Desenvolvimento Back-end. Node.js. Banco de Dados.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Para a conclusão e obtenção do título de Bacharel em Sistemas de Informação na UFLA, um dos requisitos é a aprovação do Trabalho de Conclusão de Curso. Esse trabalho pode ser apresentado por meio de diferentes modelos, sendo um desses o Relatório de Estágio, em caso de realização do estágio em uma organização por parte do discente.

O estágio profissional é uma das possibilidades para a aplicação dos conhecimentos adquiridos durante o período acadêmico. Este relatório apresenta atividades aplicadas e conhecimentos obtidos durante um ano em uma empresa de tecnologia, desde o processo seletivo iniciado no mês de fevereiro de 2021 até a finalização do estágio em fevereiro de 2022.

A oportunidade oferecida pela organização possibilitou o primeiro contato com o ambiente profissional e forneceu aprendizados que apenas são adquiridos por meio de prática em um ambiente organizacional real.

Neste capítulo serão apresentados uma visão geral da organização, os objetivos e a estrutura do documento.

1.1 Objetivos

O objetivo deste documento é abordar a experiência e as atividades realizadas durante um ano de trabalhos na área de desenvolvimento back-end, especificamente utilizado "Node.js", na empresa *ioasys*.

O estágio teve como propósito executar as atividades de levantamento e análise de requisitos, escolhas das melhores tecnologias a serem utilizadas nos projetos, implementação de projetos com boas práticas de desenvolvimento e a promoção de uma comunicação ativa, coerente e estruturada com clientes e equipe.

Neste relatório também serão apresentadas as ferramentas utilizadas e as principais contribuições feitas durante o período. Além disso, uma análise de como os conhecimentos adquiridos durante a formação acadêmica puderam contribuir para o desenvolvimento profissional e das atividades no ambiente de trabalho.

1.2 Sobre a empresa

Criada em 2012, a *ioasys* (Innovation Oasys Desenvolvimento de Sistemas) é uma empresa de inovação tecnológica, com sede em Belo Horizonte, no estado de Minas Gerais e que emprega mais de 400 colaboradores. Além do escritório sede, conta com um escritório em São Paulo e Lavras. A organização está listada pelo ranking GPTW[1] de 2022 entre as melhores empresas de médio porte para trabalhar em Minas Gerais, reconhecida pelo trabalho com metodologias ágeis e pela cultura organizacional: "de pessoas para pessoas". A área de atuação da empresa é no desenvolvimento de softwares, ofertando carreiras de desenvolvedor e de todas as áreas responsáveis por a criação de um sistema. No portfólio[2] da organização estão registrados trabalhos com grandes empresas, como Banco Inter, Localiza e VLI.

Para ingressar como estagiário na organização foi necessário participar do "Camp", um programa realizado anualmente pela companhia para formação de pessoas, que no final do processo podem ser premiadas com recompensas, como uma vaga de estagiário. São ofertadas diversas trilhas nas áreas de design, projetos, tecnologia e inovação, sendo a trilha escolhida pelo estagiário a de "Desenvolvimento back-end Node.js".

Todo estagiário na *ioasys* possui um mentor para auxiliar nas tarefas e no desenvolvimento técnico, além disso, é feito um acompanhamento muito constante por parte do departamento de pessoas, para receber as necessidades do estagiário e também esclarecer como está o relacionamento com os outros funcionários e a adaptação ao ambiente da empresa.

Além dos trabalhos realizados com clientes externos, a *ioasys* conta com projetos internos da organização. Esses projetos possibilitam aos novos colaboradores se adaptarem ao ambiente organizacional e desenvolver técnicas, da respectiva área e de comunicação. Durante a trajetória de um ano de trabalhos foi possível vivenciar experiências no desenvolvimento de um projeto interno e um projeto externo, no trabalho com uma API para o aplicativo de uma empresa de tintas, que serão apresentados em detalhes no Capítulo 3.

1.3 Estrutura do documento

O restante deste documento está organizado da seguinte forma: no terceiro capítulo, é apresentado como foi a participação no programa para ingressar na organização e são descritos,

[1] Disponível em <https://gptw.com.br/ranking/melhores-empresas/>. Acesso em 6 fev. 2023.
[2] Disponível em <https://ioasys.com.br/>. Acesso em 12 mar. 2023.

de forma detalhada, os projetos e as principais atividades realizadas como desenvolvedor back-end, a estrutura hierárquica da empresa da perspectiva do colaborador e a descrição do período de adaptação com o mentor técnico definido pela empresa; e, no capítulo quatro encontra-se a conclusão e como a participação nos projetos, com diversidade de ferramentas, contribui para carreira profissional de um desenvolvedor iniciante, estagiário ou júnior

2 REFERENCIAL TEÓRICO

Este capítulo apresenta as ferramentas e técnicas utilizadas, durante o período de trabalhos, para o desenvolvimento dos projetos. Contém as definições das tecnologias e também as ferramentas referentes a metodologias ágeis, a desenvolvimento de *software*, a gerenciamento de banco de dados, a infraestrutura, entre outros.

2.1 Metodologias ágeis

Diferente das formas de gestão de projetos para desenvolvimento de *software* mais utilizadas no século passado, as metodologias ágeis se popularizaram a partir do manifesto ágil, que reuniu especialistas e criadores de diferentes métodos para desenvolvimento de *software*, criando a Aliança Ágil e gerando diretrizes a serem seguidas por esses métodos. Os quatro principais conceitos que norteiam o manifesto ágil estão demonstrados na página do movimento (BECK et al., 2001):

1. Indivíduos e interações: mais que processos e ferramentas.
2. *Software* em funcionamento: mais que documentação abrangente.
3. Colaboração com o cliente: mais que negociação de contratos.
4. Responder a mudanças: mais que seguir um plano.

Apesar das características apresentadas, de uma rápida adaptação e a priorização do *software* funcionando, essas metodologias não ignoram a utilização de documentação, de processos ou ferramentas, apenas privilegia os citados anteriormente. Existem diversas metodologias ágeis utilizadas atualmente no mercado de desenvolvimento de *software*, entre essas estão o Scrum e o Kanban que serão apresentados a seguir.

2.1.1 Scrum

O Scrum é um *framework* classificado como uma metodologia ágil e que é amplamente utilizado por empresas da área de desenvolvimento de *software* atualmente. Foi criado por Jeff Sutherland e Ken Schwaber, tem como característica principal a capacidade de adaptação na solução dos problemas e segundo o Guia do Scrum, o *framework* tem como base o empirismo e

lean thinking, ou seja, a prática e a concentração no essencial (SCHWABER; SUTHERLAND, 2020).

Para a aplicação da metodologia, é necessário um grupo de no máximo dez pessoas e cada integrante tem um papel dentro desse conjunto, chamado de *Scrum Team*. As possíveis responsabilidades dentro do time são:

- *Developers*: que trabalham e são responsáveis pela criação do produto, através da execução das atividades.
- *Product Owner*: tem a função de maximizar e garantir o valor do produto entregue pelo Scrum Team.
- *Scrum Master*: responsável por aplicar o *framework* de formar correta e atuar como líder dentro do time.

Alguns ritos devem ser seguidos pelo time para que o Scrum funcione e para que seus principais pilares sejam aplicados: transparência, inspeção e adaptação. O primeiro evento é a *Sprint*, que tem duração de um mês ou menos e todo o trabalho para atingir a meta, incluindo os outros ritos, acontecem dentro desse período. A *Sprint Planning* é o rito de início de uma *Sprint*, nessa etapa são discutidos e definidos os itens propostos pelo *Product Owner*. Outro evento é a *Daily Scrum*, que é um evento diário utilizado para atualização e inspeção do progresso da meta da *Sprint*.

No final da Sprint são realizadas a *Sprint Review* e a *Sprint Retrospective*, na primeira é feita uma análise sobre o resultado do período de trabalho e determinadas possíveis melhorias ou mudanças e na Retrospective é feita uma discussão sobre maneiras de aumentar a performance da Sprint.

Além das funções e eventos, outra propriedade do *framework* são os artefatos, que são responsáveis por representar os planos e trabalhos, permitindo a transparência e adaptações. Existem três artefatos e cada um com a sua funcionalidade:

- *Product Backlog*: lista de funcionalidades que serão implementadas e necessárias para a melhoria do produto, definindo a meta do produto.
- *Sprint Backlog* é a lista de tarefas feitas por e para os desenvolvedores, que as implementarão durante a *Sprint*.

- Incrementos: são pequenas partes de trabalho que somadas fornecem valor ao produto, essas podem ser entregues diversas vezes durante uma *Sprint*.

Embora existam regras para a aplicação do Scrum, muitas organizações e times utilizam o *framework* com algumas modificações e combinando mais de uma metodologia ágil no desenvolvimento de um projeto. Apesar das alterações aplicadas, é necessário respeitar as diretrizes do Ágil e seus conceitos, quanto as prioridades durante a criação de um sistema.

2.1.2 Kanban

Utilizado atualmente no desenvolvimento de *softwares*, a palavra *kanban* tem origem japonesa (que significa "cartão visual") e já era utilizada para se referir ao processo de produção *just-in-time* utilizado nas fábricas. O método Kanban foi utilizado pela primeira vez no processo de criação de *software* pelo David Anderson e utiliza um quadro que é dividido por colunas, em que são inseridas histórias de usuários para que sejam transformadas em funcionalidades.

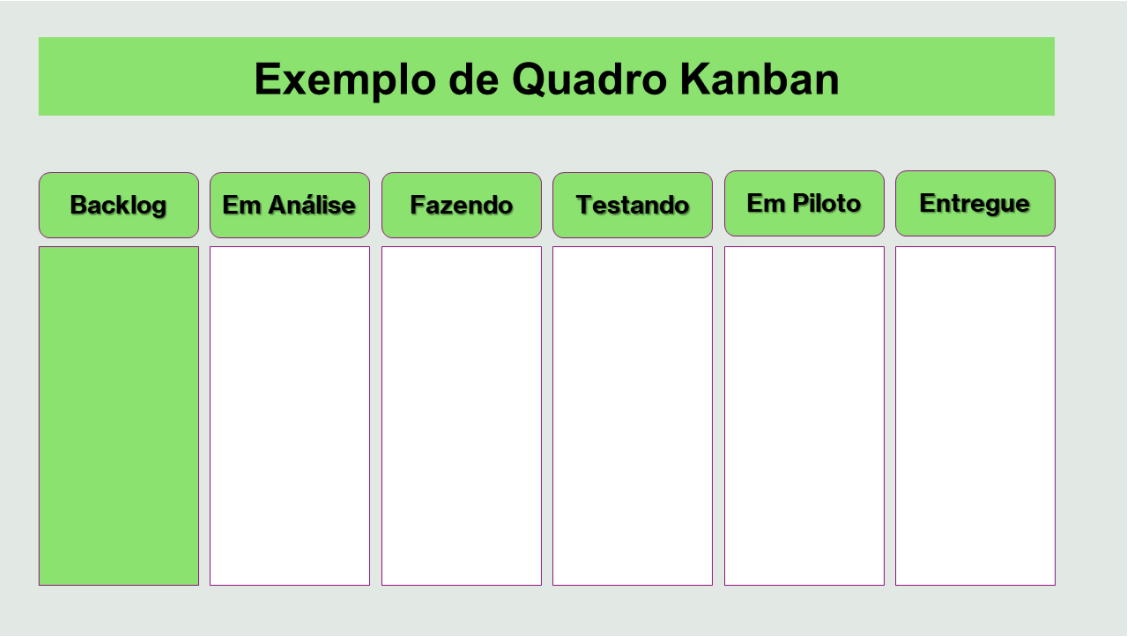
Como apresentado na Figura 2.1, apesar de algumas variações, a primeira coluna do quadro é referente aos itens de *backlog* e as seguintes referentes aos trabalhos de desenvolvimento, validação do código e outros possíveis status definidos pelo time. Muitas vezes é utilizada uma ordem para execução das atividades, sendo as colocadas na parte superior do painel mais importantes que as inferiores.

Uma aplicação que pode ser feita do Kanban é usá-lo junto com o *framework* Scrum (Seção 2.1.1) para complementar a implementação de ambos. A utilização do quadro não invalida a utilização da metodologia, isso fica explícito no Guia Kanban para Scrum Teams:

> Quando as práticas Kanban são aplicadas ao Scrum, elas promovem um foco em melhorar o fluxo por meio do ciclo de feedback; aumentando a transparência e a frequência de inspeção e adaptação tanto para o produto quanto para o processo.(VACANTI; YERET, 2021, p. 4)

Dessa forma, a aplicação do quadro Kanban durante os ritos do Scrum é importante, principalmente para a organização das funcionalidades que devem ser implementadas, para uma melhor visualização por parte dos desenvolvedores, criando uma ordem de importância para as tarefas, e também para ser mais um canal de comunicação entre o time.

Figura 2.1 – Exemplo de quadro Kanban



Fonte: Sabino (2023)

2.2 Tecnologias e Ferramentas de Desenvolvimento Back-end

Diferente da camada front-end, o back-end é a parte de um código ou aplicação que não pode ser acessado pelo usuário e contém qualquer funcionalidade que necessita ser acessada por recursos digitais (SOUZA; LIMA; CARIDADE, 2022). Diversas linguagens de programação podem ser utilizadas para a criação de aplicações back-end, como o JavaScript com o ambiente fornecido pelo Node.js (NETO, 2020). Essas aplicações podem se comunicar com diversos recursos, que auxiliam no gerenciamento dos dados, de arquivos e de segurança para o projeto.

Esta seção apresenta a linguagem utilizada, as estruturas de comunicação, as ferramentas empregadas para o desenvolvimento de aplicações back-end, as tecnologias utilizadas para o gerenciamento e interação com os bancos de dados, o padrão de documentação e o sistema de controle de versão utilizado durante o período do estágio.

2.2.1 JavaScript

JavaScript é uma linguagem web, baseada em objetos, interpretada, não tipada e inicialmente a linguagem foi projetada para executar nos navegadores, ou seja, no lado do usuário. Embora utilize o termo “Java” em seu nome, as linguagens se diferenciam muito, tendo apenas uma pequena semelhança sintática.

A principal funcionalidade que gerou a criação da linguagem, foi a necessidade de uma linguagem *client-side*[1], para atender a crescente demanda de páginas web que se comportassem de forma dinâmica, ou seja, gerando efeitos visuais na interação do usuário com a página, que anteriormente eram mostradas sem nenhuma ou com pouca interação. Na Figura 2.2 é apresentado um exemplo do uso da função "alert" da linguagem JavaScript incorporada diretamente no arquivo HTML[2]. Essa função é responsável por mostrar ao usuário uma mensagem seguida de um botão de confirmação.

Figura 2.2 – Função "alert"na linguagem JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>Olá mundo com JavaScript! </title>
  <script>
    alert('Olá Mundo!');
  </script>
</head>
</html>
```

Fonte: Autor

Como é explicado por Flanagan (2004), para trabalhar com vetores, textos, datas e expressões regulares, o JavaScript utiliza uma API básica, sem funcionalidade de entrada ou saída, que fica a cargo da plataforma hospedeira, que muitas vezes é o próprio navegador web.

2.2.2 Node.js

Atualmente a linguagem JavaScript (Seção 2.2.1) pode ser utilizada no desenvolvimento back-end *server-side*[3], não somente com a função de gerar interação com elementos da página web. De acordo com Neto (2020), quem transformou esse cenário foi o Node.js, sendo utilizado por grandes empresas que recebem muitas requisições diariamente.

Node.js é um ambiente de execução de códigos JavaScript e funciona em single-thread, ou seja, apenas uma thread executa o processo. Para executar a linguagem, o Node.js utiliza o interpretador V8 do Google Chrome fora do navegador e para trabalhar com apenas uma thread,

[1] Client-side é o termo utilizado para se referir a uma aplicação que é executada no ambiente do cliente, ou seja, do lado do usuário.

[2] HTML (Linguagem de Marcação de HiperTexto) é uma linguagem utilizada para a criação de páginas Web.

[3] Server-side é utilizado para identificar aplicações que executam no ambiente do servidor.

essas aplicações utilizam de um recurso chamado event-loop, que trabalha de forma assíncrona e sem bloqueio. Na Figura 2.3, apresenta-se um trecho de código que executa um servidor HTTP. Ao ser iniciado, esse servidor espera uma chamada no endereço “localhost”, especificamente na porta 3000. Após essa requisição ser feita é retornado o texto “Olá Mundo!”.

Figura 2.3 – “Olá Mundo!” utilizando Node.js

```
const http = require('http');

const servidor = http.createServer((req, res) => {
  res.writeHeader(200, {'Content-Type': 'text/plain; charset=utf-8'});
  res.end('Olá Mundo!');
})

const porta = 3000;
servidor.listen(porta, 'localhost', () => {
  console.log(`Servidor executando na porta: ${porta}`);
})
```

Fonte: Autor

Considerando que um evento é feito para a aplicação, a resposta dessa tarefa não é esperada, sendo chamado outro evento que está na fila. Sendo assim, várias solicitações podem ser colocadas na fila ao mesmo tempo, sem esperar que uma solicitação seja concluída.

2.2.3 Node Package Manager

O NPM (Node Package Manager) é um repositório e gerenciador de pacotes para o Node.js (Seção 2.2.2), por meio do qual é possível fazer o controle de versões de pacotes disponíveis para o ambiente e que é necessário para executar um determinado projeto (NPM, 2023). Todo o trabalho com as dependências é feito de maneira simples, apenas executando um comando para adicionar ou remover um pacote do projeto.

Utilizando um arquivo do tipo JSON[4], chamado *package*, o NPM faz o registro de todas as informações da aplicação. Dessa forma, o gerenciador consegue estabelecer quais são as dependências necessárias para que a aplicação seja executada. Para se iniciar um projeto utilizando o NPM, após instalá-lo na máquina, é necessário apenas executar o comando “npm init” no terminal do sistema operacional, desse modo, é gerado o arquivo *package*, que é demonstrado na Figura 2.4.

[4] JSON (JavaScript Object Notation) é um formato de arquivo utilizado para estruturar dados em forma de texto para transferência de dados.

Figura 2.4 – Arquivo "package.json"

```
{
  "name": "exemplo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Pedro",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Fonte: Autor

Mesmo sendo atualmente o gerenciador de pacotes padrão do Node.js, o NPM tem concorrentes, sendo o maior desses o *yarn*[5], que tem os comandos bem parecidos com o gerenciador padrão. A principal ideia na criação do *yarn* foi criar um gerenciador mais ágil e seguro que o NPM, mas ambos são usados atualmente no mercado de desenvolvimento.

2.2.4 TypeScript

A Microsoft, com base no JavaScript, desenvolveu um superconjunto para tornar mais eficiente e fornecer recursos que não existiam na linguagem inicial, como tipagem estática, orientação a objetos e facilidade para o desenvolvedor encontrar erros. Para utilizar a linguagem, o pacote deve ser instalado utilizando o NPM (Seção 2.2.3) ou utilizando a IDE[6] Visual Studio, que também foi criada pela Microsoft (TYPESCRIPT, 2022).

A tipagem na linguagem pode ser feita de forma implícita e explícita. A primeira acontece quando o valor de uma determinada variável é definido na declaração e no segundo caso, ocorre quando o tipo é definido, não o valor. Em ambos os casos, o tipo da variável não pode ser alterado depois de ser definido, caso ocorra, um erro será gerado. Na Figura 2.5 são mostrados exemplos de tipagens utilizando o TypeScript e os erros gerados ao atribuir um tipo diferente do declarado a uma variável.

[5] Disponível em <https://yarnpkg.com/>. Acesso em 6 fev. 2023.

[6] Um ambiente de desenvolvimento integrado (IDE) é um sistema utilizado para o desenvolvimento de *softwares* e conta com ferramenta que facilitam esse processo.

Figura 2.5 – Tipagem no TypeScript

```
1   var implicito = 'Olá Mundo!'; //Type String
2   implicito = 'Novo Olá Mundo!';
3   implicito = 1; //Type 'number' is not assignable to type 'string'.
4
5   var explicito: string; //Type String
6   explicito = 'Olá Mundo!'
7   explicito = 1; //Type 'number' is not assignable to type 'string'.
8
```

Fonte: Autor

Apesar de ser considerada uma linguagem e ter todas as funcionalidades do JavaScript, pela própria documentação, o TypeScript pode ser considerado apenas um subconjunto com adaptações. O que faz muitos desenvolvedores não considerá-lo uma linguagem é o fato de ser um código "transpilado": quando um projeto TypeScript é compilado, o código é convertido para JavaScript antes de ser executado, um processo que é conhecido como "transpilação".

2.2.5 API REST

Para entender o conceito de API REST é necessário entender o que significa a sigla API e a arquitetura REST. API é a sigla de Interface de Programação de Aplicações e funciona como uma ponte, com um conjunto de funções e definições, para que um sistema acesse recursos que estão alocados em outro determinado sistema. Para que essa comunicação seja realizada, um serviço não precisa saber como foi implementado e tratada os recursos desejados. Uma API pode ser pública, privada, entre parceiros ou composta (AWS, 2023b):

- Pública: é classificada quando uma API é aberta para acesso e pode ser utilizada por qualquer aplicações. Exemplos comuns são APIs de autenticação de usuário e a API que manipula informações sobre o Código de Endereçamento Postal (CEP)[7].
- Privada: são interfaces utilizadas internamente em projetos de organizações, ou seja, apenas os sistemas da empresa podem acessar os recursos.
- Entre parceiros: são muito utilizadas quando organizações parceiras desejam compartilhar informações ou funcionalidades com outras empresas.

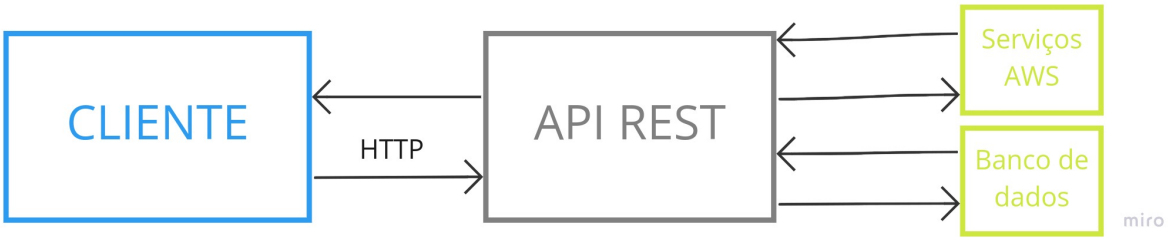
[7] Disponível em <https://www.gov.br/conecta/catalogo/apis/cep-codigo-de-enderecamento-postal/>. Acesso em 6 fev. 2023.

- Composta: são interfaces que combinam uma ou mais APIs diferentes para resolver problemas difíceis.

Quando se deseja acessar algum recurso fornecido por uma API, é necessário que a requisição seja feita de maneira correta, ou seja, utilizando um padrão para que a interface entenda qual o recurso desejado pelo cliente. Para que isso aconteça, a documentação da API informa como devem ser feitas as solicitações e também deixa explicitado como será o padrão das respostas retornadas. Esses recursos que são manipulados e devolvidos nas respostas podem ser de todos os tipos de dados possíveis, como texto, imagem, localização, entre outros.

A sigla REST representa Transferência de Estado Representacional e é um estilo arquitetural que define normas utilizadas por determinadas aplicações. Nessa arquitetura, as aplicações utilizam o HTTP[8] e as funções presentes nesse protocolo, como POST, DELETE e GET, para realizar a comunicação e manipulação dos dados. As aplicações que seguem os padrões definidos pela arquitetura REST, são denominadas "RESTful". Na Figura 2.6 é demonstrado o fluxo de uma API REST que recebe a demanda de um cliente, faz a interação com serviços externos e devolve a resposta para operação demandada.

Figura 2.6 – Diagrama de API REST



Fonte: Autor

As APIs REST, além de seguirem um padrão de implementação tem como característica a ausência de estado, isto é, o servidor não armazena informações do cliente. Sendo assim, a cada pedido realizado pelo cliente ao servidor, é necessário que sejam enviadas as informações para que a resposta seja a esperada.

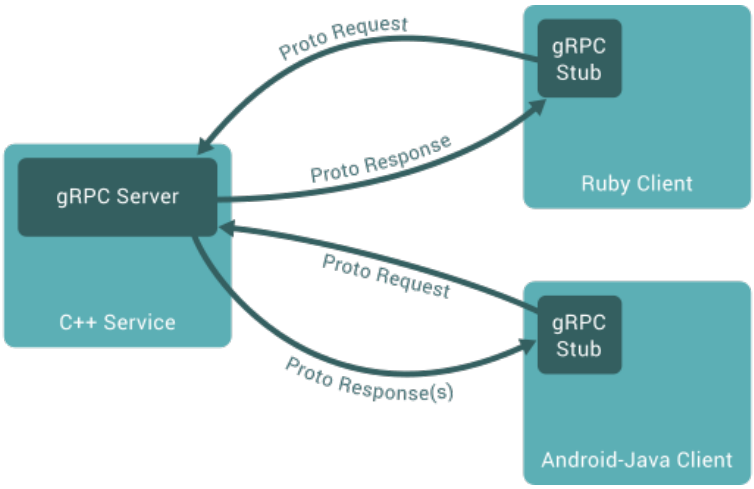
Uma vantagem para o uso desse tipo de interface é a independência entre a linguagem utilizada no cliente e no servidor, podendo cada uma das aplicações ser desenvolvida utilizando uma tecnologia (AWS, 2023a). Dessa forma, uma API pode comunicar com uma aplicação *Web* e também com uma *Mobile*, fornecendo os mesmos recursos para ambas.

[8] HTTP (Hypertext Transfer Protocol) é um protocolo para comunicação cliente-servidor que utiliza hipertexto para fazer comunicação entre as partes.

2.2.6 gRPC

O gRPC é uma estrutura RPC (Chamada de Procedimento Remoto) que foi desenvolvida pela Google. Tem a função de fornecer para uma aplicação (cliente) a possibilidade de acessar diretamente um método presente em outro aplicativo (servidor), facilitando a comunicação entre serviços (GRPC, 2022). Diferente da comunicação utilizada pelas APIs REST (Seção 2.2.5), o gRPC utiliza buffers de protocolo para a comunicação entre os servidores. Os arquivos utilizados para definir essa estrutura de dados são da extensão "proto", que isoladamente não contém nenhuma funcionalidade.

Figura 2.7 – Comunicação gRPC com dois clientes



Fonte: gRPC (2022)

A Figura 2.7 apresenta duas aplicações clientes, uma na linguagem Ruby e outra na linguagem Java (Android), que estão consumindo os métodos do servidor, na linguagem C++. Essa comunicação direta pode ser feita através do *stub*, um objeto que representa as funcionalidades do servidor.

Além da utilização de *buffers* de protocolo, o gRPC utiliza o protocolo HTTP/2, que consegue enviar mais de uma requisição para o servidor e receber mais de uma resposta, em uma mesma conexão. Segundo Oliveira (2021), todas essas características contribuem para que o gRPC consiga ser mais rápido que o REST, contudo gera maior esforço para ser implementado, devido a definição dos *buffers* de protocolo. Dessa forma, é necessário verificar em qual contexto se deseja aplicar cada um dos protocolos, ou seja, tendo como preferência o que melhor se enquadre nas características do projeto.

2.2.7 PostgreSQL

Popularmente conhecido como "Postgres", é um sistema gerenciador de bancos de dados (SGBD) objeto-relacional de código aberto e de acordo com a própria documentação mais atualizada, o sistema visa estar em conformidade com o padrão SQL, que é definido pela ISO/IEC 9075 (POSTGRESQL, 2022). Além de ser um sistema de código aberto, que facilita a manutenção, outras características que favorecem a utilização do SGBD são:

1. Suporte a uma grande quantidade de dados nativamente, além de conceder a possibilidade da criação de tipos personalizados e de extensões.

2. Por ser desenvolvido e mantido por muito tempo, desde 1986, é um sistema muito tolerante a falhas.

3. Tem uma ótima escalabilidade, suportando uma grande quantidade de dados gerenciados e também de usuários conectados simultaneamente.

4. O sistema tem suporte para uma grande quantidade de linguagens de programação, como Java, Ruby e Go, evitando conflitos.

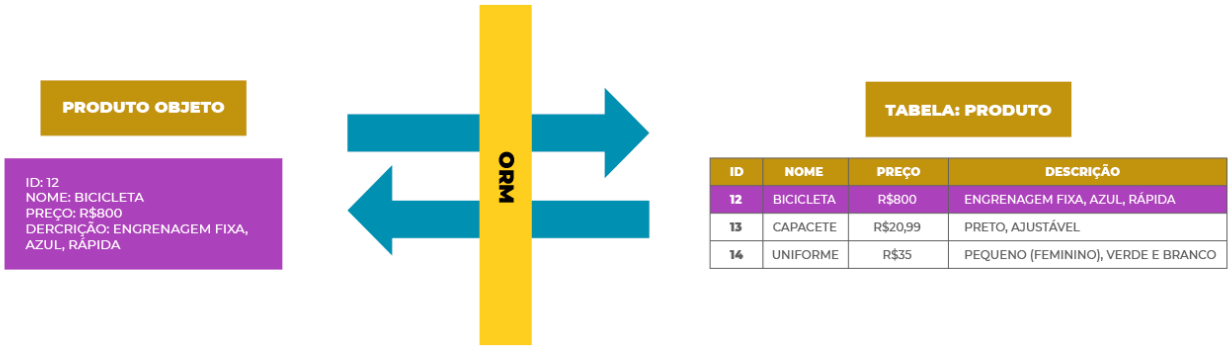
2.2.8 ORM

Um ORM (*Object Relational Mapper*) é utilizado para facilitar a interação entre uma aplicação orientada a objetos com um banco de dados relacional, através de transformações entre tabelas e objetos de uma determinada linguagem de programação. Diante disso, a utilização de um ORM reduz o problema da impedância objeto-relacional, que ocorre quando uma aplicação orientada a objetos utiliza um banco de dados relacionais, ou seja, um conflito entre dois paradigmas (PLÁCIDO; CUNHA, 2008).

A Figura 2.8 apresenta um exemplo de comunicação entre uma tabela de “Produtos” e um objeto do tipo “Produto” em uma determinada linguagem orientada a objetos. O ORM tem a função de converter o objeto para o padrão do banco de dados, em operações de entrada, e para as operações de saída de dados é feito o fluxo contrário.

Existem dois tipos de padrões utilizados em ORMs, o *Active Record* e o *Data Mapper*. No primeiro, as funções para acessar o banco de dados ficam dentro da classe que simboliza

Figura 2.8 – Fluxo de um ORM



Fonte: Fonseca (2019)

a tabela na aplicação, muitas vezes os métodos são herdados de uma classe padrão. No padrão *Data Mapper* é utilizada uma classe chamada de Repositório, específica para realizar a implementação desses métodos.

2.2.9 TypeORM

TypeORM é um *framework* com suporte para o desenvolvimento Node.js (Seção 2.2.2) e outras plataformas, que utiliza TypeScript (Seção 2.2.4) e JavaScript (Seção 2.2.1) para ser implementado. Esse *framework* suporta os dois padrões utilizados para a implementação de um ORM, o *Active Record* e o *Data Mapper*.

Uma das propriedades utilizadas é chamada de *Entity*, essa classe é responsável por definir o modelo da tabela do banco de dados e através dela é possível gerar os arquivos chamados de *migrations*. As *migrations* são essenciais na utilização de um ORM, tem a função de executar e gerenciar o versionamento das modificações estruturais no banco de dados, como a alteração do nome de tabelas, exclusão de atributos, criação de relacionamento entre tabelas, entre outras. No TypeORM, as *migrations* podem ser geradas automaticamente através das modificações nas entidades ou implementadas separadamente, ambas as opções são feitas através de uma interface de linha de comando disponibilizada pelo *framework* (TYPEORM, 2023).

A Figura 2.9 apresenta um exemplo de classe que é utilizada para definir o modelo de uma tabela do banco de dados. O valor “tasks” utilizado como parâmetro na propriedade *Entity* refere-se ao nome da tabela no banco. A propriedade *PrimaryGeneratedColumn* é utilizada para identificar a chave primária da tabela. Por fim, os três decorators que são finalizados com

o termo “DateColumn” têm a função de definir os atributos que armazenam os valores de data e hora de criação, atualização e deleção de um determinado valor na tabela.

Figura 2.9 – Modelo de classe "tarefas"no TypeORM

```
import {
  Column,
  CreateDateColumn,
  DeleteDateColumn,
  Entity,
  PrimaryGeneratedColumn,
  UpdateDateColumn
} from 'typeorm';

@Entity('tasks')
export class Task {
  @PrimaryGeneratedColumn('uuid')
  public id: string;

  @Column()
  public description: string;

  @Column()
  public period: Date;

  @Column()
  public completed: boolean;

  @CreateDateColumn({ name: 'created_at' })
  public createdAt: Date;

  @UpdateDateColumn({ name: 'updated_at' })
  public updatedAt: Date;

  @DeleteDateColumn({ name: 'deleted_at' })
  public deletedAt: Date;
}
```

Fonte: Autor

2.2.10 NestJS

NestJS é um *framework* de desenvolvimento back-end que utiliza Node.js e tem como linguagem padrão o TypeScript, além disso, utiliza elementos de Programação Orientada a Objetos, Programação Funcional e Programação Reativa Funcional. Internamente o NestJS utiliza o Express que é um *framework* robusto para trabalhar com o HTTP, apesar disso, tem um alto nível de abstração, o que facilita o desenvolvimento.

O NestJS é modularizado, ou seja, cada aplicação possui pelo menos um módulo, que é o módulo principal. Além dos módulos, o *framework* utiliza algumas estruturas que garantem o seu funcionamento, duas dessas estruturas são os *Controllers* e os *Providers* (NESTJS, 2017).

Os *controllers* têm a responsabilidade de tratar as requisições feitas pelo cliente e fornecer uma resposta. Cada controlador pode ter uma ou mais rotas definidas na classe e o mecanismo de roteamento do *framework* que define para qual rota foi feita a requisição, isso é possível pelos prefixos definidos nos *decorators*. Além do *decorator* "Controller", que pode receber um prefixo, cada rota é especificada por um *decorator* com método HTTP.

Os *providers* são estruturas que são injetadas como uma dependência, dessa maneira diminui o acoplamento entre as classes. Por se tratar de uma dependência, os provedores podem ser classificados de diferentes maneiras, entre elas:

- *Services*: geralmente são responsáveis por realizar toda regra de negócio executada por uma rota, fazendo validações, comunicação com os repositórios e com provedores externos, como comunicação com outra API.

- *Repositories*: são as classes utilizadas para fazer a comunicação direta com o banco de dados, quando está se utilizando um ORM com o padrão *Data Mapper*. O NestJS fornece um pacote do TypeORM (Seção 2.2.9), para utilizá-lo é necessário instalar as dependências.

Como mostra a Figura 2.10, um módulo pode conter alguns tipos de propriedades, que são: *providers*, *controllers*, *imports* e *exports*. Esse exemplo é referente a um módulo de “tarefas” e as rotas estão separadas por arquivos, por isso existe um controlador e serviço para cada uma delas. Outro detalhe é o uso do módulo do TypeORM (Seção 2.2.9) oferecido pelo NestJS (TypeOrmModule), em que está sendo importado o único repositório (TaskRepository) usado nesse módulo.

Figura 2.10 – Módulo "tarefas"no *framework* NestJS

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { TaskRepository } from 'shared/repositories/task.repository';
import { CreateTaskController } from './context/create/create.controller';
import { CreateTaskService } from './context/create/create.service';
import { DeleteTaskController } from './context/delete/delete.controller';
import { DeleteTaskService } from './context/delete/delete.service';
import { FindTaskController } from './context/find/find.controller';
import { FindTaskService } from './context/find/find.service';
import { ListTaskController } from './context/list/list.controller';
import { ListTaskService } from './context/list/list.service';
import { UpdateTaskController } from './context/update/update.controller';
import { UpdateTaskService } from './context/update/update.service';

@Module({
  imports: [TypeOrmModule.forFeature([TaskRepository])],
  controllers: [
    CreateTaskController,
    DeleteTaskController,
    FindTaskController,
    ListTaskController,
    UpdateTaskController
  ],
  exports: [],
  providers: [
    CreateTaskService,
    DeleteTaskService,
    FindTaskService,
    ListTaskService,
    UpdateTaskService
  ],
})
export class TaskModule {}
```

Fonte: Autor

Nos *imports* são adicionados os módulos que exportam os provedores necessários para a execução desse módulo importador e nos *exports* os provedores que serão expostos. Os *controllers* recebem a lista das estruturas instanciadas junto com o módulo e os *providers* guardam as estruturas que serão injetadas.

Outra estrutura importante, que é utilizada para verificar se os usuários têm autorização para acessar determinada rota, é chamada de *Guard*. Essas classes também são injetáveis como dependências, diferente das aplicações que utilizam apenas Express em que são tratadas como *middleware*.

2.2.11 JSON Web Token

Para realizar autenticação de um usuário é preciso utilizar uma chave e, para isso, são utilizados padrões. O JWT é um desses padrões, que além de ser usado para autenticar, também é aplicado para troca de informações e, além disso, é uma técnica definida pela RCF 7519 [9] (MONTANHEIRO; CARVALHO; RODRIGUES, 2017). O *token* utiliza Base64[10] e consegue armazenar informações utilizando JSON.

Os *tokens* podem ser criptografados, quando ocultam as informações contidas neles, não sendo possível verificar a integridade e também podem ser assinados, quando é utilizado um segredo ou um par de chaves (pública e privada) para certificar a integridade das informações contidas no JWT.

A Figura 2.11 apresenta um JWT composto pelo *header*, *payload* e pela *signature*. O *header* é composto pelo algoritmo de assinatura e pelo tipo do *token*. O *payload* é um objeto com informações de configuração, como tempo de expiração, e atributos usados nas aplicações. Por fim, a *signature* é a associação dos valores codificados do *header* e do *payload*, juntamente com uma chave para verificar a integridade da mensagem.

2.2.12 AWS

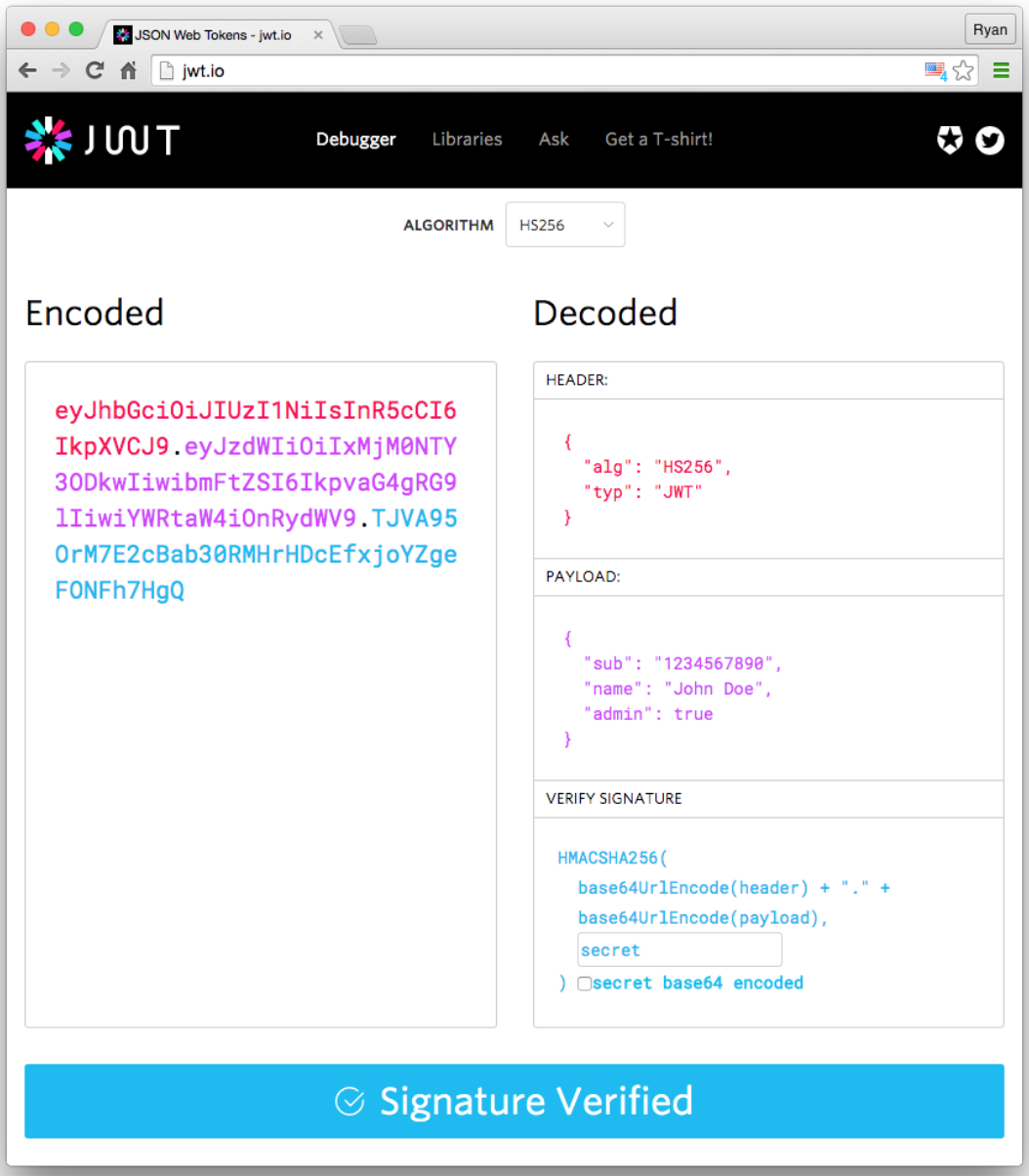
AWS é a sigla referente a Amazon Web Services e corresponde a uma arquitetura IaaS (infraestrutura como serviço) que oferece algumas alternativas PaaS (plataforma como serviço) (VERAS, 2013). Segundo Lecheta (2014), três pontos principais são os necessários para caracterizar a plataforma:

1. Inicialmente é possível utilizar gratuitamente.
2. Em pouco tempo é possível executar máquinas virtuais e aplicativos em minutos.
3. Será pago apenas pelos serviços utilizados.

A plataforma oferece um painel[11] em que é possível gerenciar os mais de 200 serviços ofertados através de uma interface web, sendo algumas dessas funções:

[9] Disponível em <https://www.rfc-editor.org/rfc/rfc7519>. Acesso em 6 fev. 2023.
[10] Base64 é uma forma codificação utilizada para transferência de dados que é constituído por 64 tipos de caracteres.
[11] Disponível em <https://console.aws.amazon.com/>. Acesso em 6 fev. 2023.

Figura 2.11 – Composição do JWT



Fonte: Auth0 (2023)

- Amazon S3 (Simple Storage Service): serviço para armazenamento de objetos. De acordo com Veras (2013), o S3 consegue ser altamente escalável e permite o acesso simultâneo para a leitura e para o salvamento dos dados por diferentes clientes ou *threads*.

- AWS Lambda: responsável por executar aplicações utilizando o modelo *serveless*[12], através uma determinada parte de código de uma linguagem de programação e sem a utiliza-

[12] No modelo *serveless* os recursos são executados sob demanda, sem a necessidade de gerenciar um servidor.

ção de um servidor. Os custos do uso do AWS Lambda são cobrados por execução, ou seja, se o código não for executado, não é cobrado. Para integrar esse serviço durante o desenvolvimento Node.js (Seção 2.2.2) é possível utilizar o *framework* Serverless[13] que é acessível pelo NPM (Seção 2.2.3).

- Amazon Cognito: a função desse serviço é o gerenciamento de usuários, autorização e autenticação. Esses usuários podem ser separados por grupos, para fornecer acesso ao aplicativo ou para fornecer acesso a outros serviços da AWS, nesse último caso chamado de grupos de identidades.

2.2.13 Swagger

O Swagger fornece um grupo de ferramentas para auxiliar desenvolvedores durante a criação de APIs. Uma dessas ferramentas, permite a especificação de uma API e a execução das rotas através de uma interface interativa, com base na especificação OpenAPI. A especificação OpenAPI fornece a possibilidade de se descobrir e entender os recursos oferecidos por uma API RESTful (Seção 2.2.5), independente da linguagem de programação que foi utilizada no desenvolvimento (SMARTBEAR, 2020). A Figura 2.12 apresenta um exemplo de uso do Swagger para documentar duas rotas de uma API de gerecimento de filmes: uma para adicionar um filme e outra para retornar uma lista.

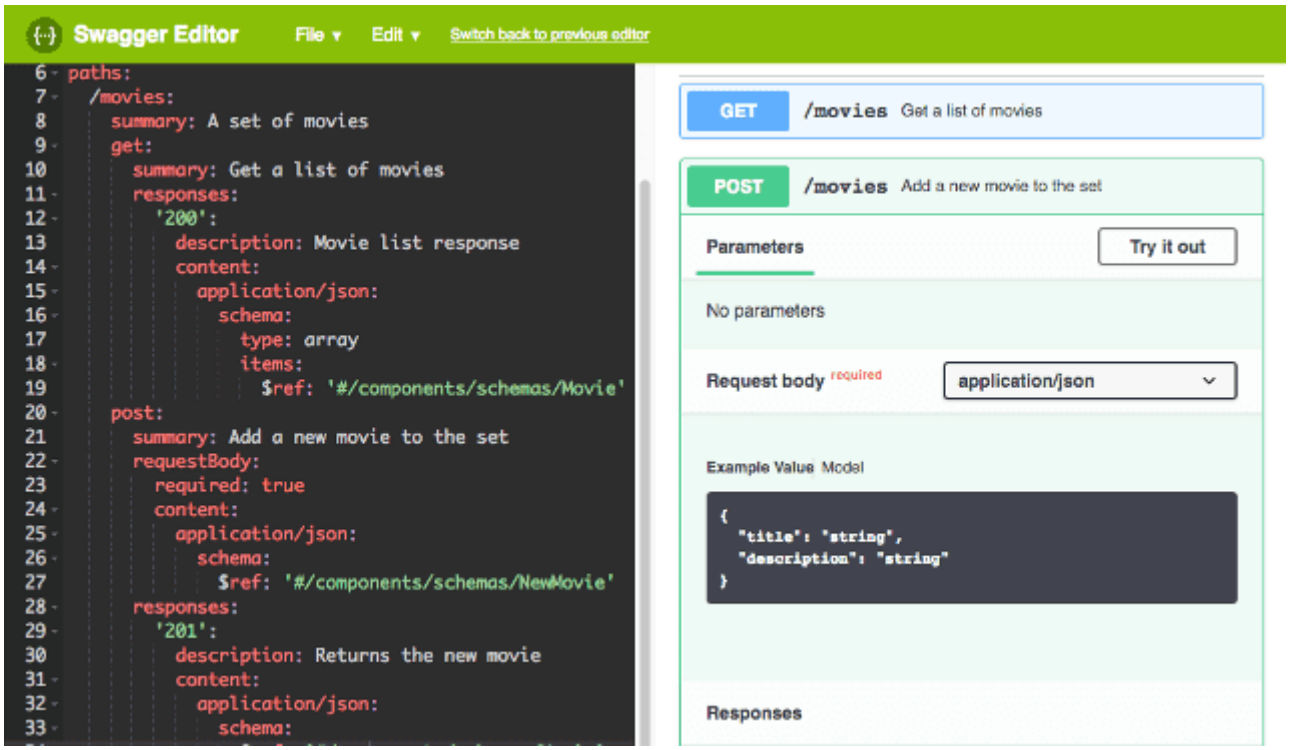
O NestJS oferece um módulo[14] para que a criação dessa interface de documentação possa ser desenvolvida através da linguagem TypeScript (Seção 2.2.4), que é utilizada pelos métodos e *decorators* presentes nesse módulo.

2.2.14 Git

O Git é um sistema de código aberto e gratuito utilizado para o controle de versão de projetos de diferentes tamanhos. Segundo Chacon e Straub (2014), o Git foi criado em 2005 por Linus Torvalds, com as metas de ser rápido, um projeto simples, com um forte suporte para o desenvolvimento não linear, ser distribuído e capaz de ser eficiente ao lidar com grandes projetos, como o núcleo do Linux[15], também criado por Linus Torvalds.

[13] Disponível em <https://www.npmjs.com/package/serverless>. Acesso em 6 fev. 2023.
[14] Disponível em <https://www.npmjs.com/package/@nestjs/swagger>. Acesso em 6 fev. 2023.
[15] Linux é um sistema operacional open source e gratuito.

Figura 2.12 – Exemplo de uso do Swagger



Fonte: Laube (2018)

Existem diversos serviços de hospedagem de projetos que utilizam o sistema Git para o versionamento, entre eles o GitHub[16] e o Bitbucket[17].

[16] Disponível em <https://github.com/>. Acesso em 6 fev. 2023.
[17] Disponível em <https://bitbucket.org/>. Acesso em 6 fev. 2023.

3 ESTÁGIO E ATIVIDADES

Este capítulo apresenta a estrutura organizacional, as atividades realizadas durante o período de estágio, a fase de aprendizados com o mentor e as tarefas realizadas em um intervalo de tempo após a finalização do estágio. Além disso, é apresentado o programa realizado anteriormente ao estágio, que foi necessário para que a vaga de estágio se realizasse.

3.1 ioasys Camp

O programa Camp é realizado há alguns anos pela *ioasys* com a intenção de encontrar novos colaboradores, para as diversas áreas de trabalhos dentro da organização. Cada área é representada por trilhas e quando um participante realiza a inscrição, deve selecionar a trilha que tem interesse em participar e realizar atividades referentes a trilha escolhida. Dessa forma, cada candidato é avaliado e os que alcançarem melhores resultados são selecionados.

Para a trilha de desenvolvimento back-end da terceira edição, que foi a escolhida pelo candidato, foram respondidos exercícios de lógica e atividades de desenvolvimento de algoritmos, essas podiam ser respondidas utilizando qualquer linguagem. Concomitante ao período em que eram analisadas as atividades respondidas pelo candidato, foi realizada uma reunião com um funcionário da organização para que o candidato fosse conhecido.

Na terceira edição o programa foi dividido em duas fases, que aconteceram entre fevereiro e junho de 2021, sendo a primeira fase referente a uma imersão técnica e na segunda fase, foram formados grupos de desenvolvimento. Todas as atividades da terceira edição do programa foram realizadas no modelo remoto. Na primeira fase foi realizado ensino de técnicas e ferramentas através de aulas que eram realizadas virtualmente. Foram realizadas quatro semanas de imersão técnica, com aulas gerais e com ensinamentos específicos de cada trilha de conhecimento.

Nas aulas gerais, todas as trilhas que estavam envolvidas no processo de desenvolvimento participavam, como as aulas que ensinavam sobre Metodologias Ágeis (Seção 2.1) e Scrum (Seção 2.1.1). Por outro lado, aconteceram aulas que não eram ministradas para todas as trilhas ou para uma única trilha, por exemplo, as apresentações de JavaScript (Seção 2.2.1) feitas para a trilha de back-end e de front-end. Dentre os assuntos apresentados para a trilha de back-end estão:

- Banco de dados relacional: teoria e modelagem utilizando um problema real.

- JavaScript: o básico e o avançado.
- Introdução ao Docker[1].
- Estrutura de projetos.

Após o encerramento das aulas da primeira fase, todos os participantes deveriam realizar um desafio referente ao tema da trilha em que estava alocado. Para os participantes da área de back-end, o desafio foi desenvolver uma API REST (Seção 2.2.5) utilizando JavaScript (Seção 2.2.1) para um serviço de informações sobre filmes, com requisições referentes ao CRUD[2] desses filmes, autenticação dos usuários, ranqueamento dos filmes por usuário, entre outras. A partir desse desafio e da participação nas aulas, foram selecionados os participantes que permaneceriam para a segunda fase.

A segunda fase foi iniciada com a divisão dos participantes em grupos, sendo cada grupo composto por no mínimo um integrante de cada trilha e liderado por um Scrum Master, que era um profissional da empresa. Cada grupo deveria criar uma solução para um problema que envolvesse um tema proposto pela empresa, desenvolver um MVP[3] e apresentar essa solução no formato de um *pitch*[4].

Por fim, depois das apresentações dos projetos, alguns destaques foram premiados e outros convidados para participar da organização, através de uma vaga de estágio ou de trabalho integral. Os projetos foram avaliados de acordo com conceito, tecnologia, inovação, viabilidade e a própria apresentação. Ao final de ambas as fases do processo foram gerados certificados pela empresa (ANEXO A).

3.2 Período de Adaptação

Na primeira semana de estágio aconteceu o *onboarding*, que é um processo para integrar um novato na empresa, apresentando os canais de comunicação, as ferramentas de gerencia-

[1] Docker é um sistema de gerenciamento de aplicações dentro de ambientes isolados chamados de containers (VITALINO; CASTRO, 2018).

[2] CRUD é a sigla utilizada para se referir as operações Create (criar), Read (ler), Update (atualizar) e Delete (deletar).

[3] MVP significa Produto Mínimo Viável, ou seja, a versão mais simples de um produto utilizado para validar uma ideia.

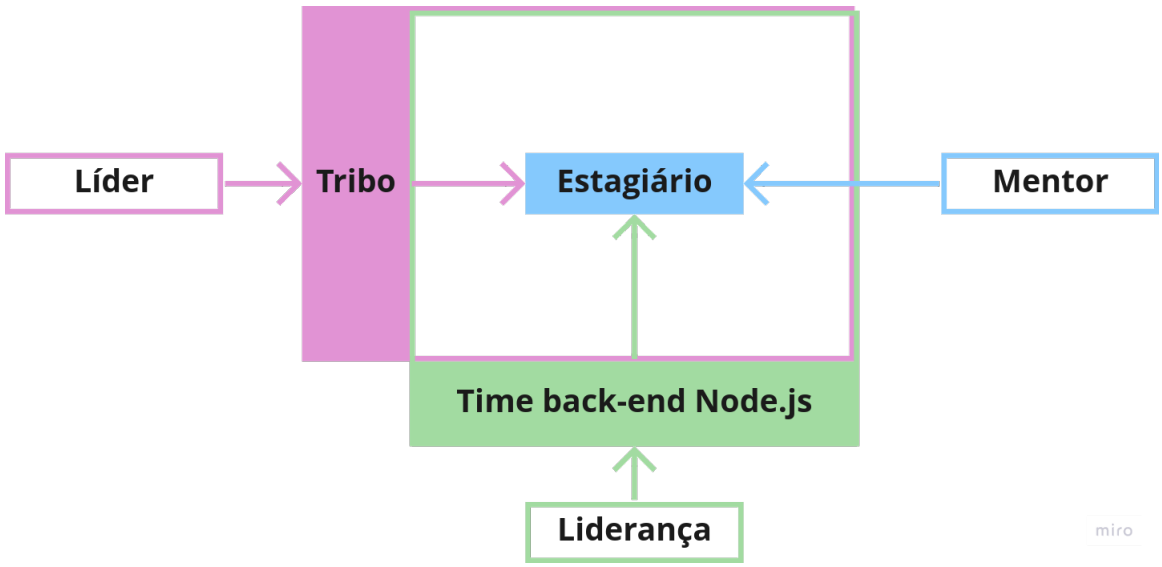
[4] Pitch é uma apresentação curta utilizada no ambiente empresarial para divulgar um projeto, que pode ser um produto ou uma empresa.

mento e o mentor técnico. Todas as atividades que aconteceram durante o período do estágio foram realizadas no modelo remoto.

Todos os estagiários e desenvolvedores júnior na *ioasys* são acompanhados por um mentor técnico, este auxilia o novo integrante da empresa na adaptação e direciona o colaborador nos estudos de novas tecnologias que são importantes para a carreira do desenvolvedor. Neste estágio, foi cobrado pelo mentor o estudo e concentração em alguns serviços da AWS (Seção 2.2.12) e no trabalho com o Docker, sendo necessário aplicar as tecnologias no projeto desenvolvido durante a segunda fase do Camp (Seção 3.1).

Durante o período de adaptação, realizado entre julho e agosto de 2021, o estagiário conheceu a estrutura organizacional da empresa e em qual grupo seria inserido. Como mostra a Figura 3.1, o estagiário participava de uma tribo (grupos responsáveis por determinados projetos) e também participava do time de back-end Node.js. Além disso, o estagiário era acompanhado diretamente pelo mentor técnico, pela liderança técnica de Node.js e também pelo líder da tribo que estava alocado.

Figura 3.1 – Estrutura organizacional ligada ao estagiário



Fonte: Autor

O mentor técnico foi responsável por decidir junto com a liderança técnica se o estagiário estava preparado para participar de um projeto. Na perspectiva do estagiário, a empresa trabalha com dois tipos de projetos. A primeira classificação de projeto é referente a sistemas da própria empresa (produtos). Diferente dos projetos internos, os externos são trabalhos realizados para clientes fora da organização, ou seja, pessoas procuram a empresa para o desenvolvimento de uma solução (serviços).

Por se tratar de um cliente interno, os projetos que são desenvolvidos para uso da empresa podem ser utilizados para alocar pessoas que estão se adaptando com o ambiente organizacional. No caso de algum problema com o desenvolvimento do projeto, a comunicação pode ser feita de forma mais rápida e evita prejudicar outro ator envolvido em um negócio externo, podendo ser prejudicada apenas a própria organização.

Neste relato, o estagiário foi alocado primeiramente no projeto de desenvolvimento de uma plataforma interna e posteriormente foi alocado em projeto que envolvia um cliente externo.

3.3 Configuração do ambiente e uso de novas tecnologias

Antes de iniciar um novo projeto é necessário que a máquina esteja configurada de acordo com as especificações pretendidas para o sistema ou em um projeto existente, as configurações necessárias. Além das especificações do projeto, cada desenvolvedor tem preferência por um tipo de tecnologia, como alguns preferem instalar todas as ferramentas utilizadas diretamente na máquina e outros desenvolvedores utilizam o Docker.

Cada projeto que o estagiário foi alocado durante o período, foi necessário verificar as tecnologias necessárias para implementar as funcionalidades requeridas. Em casos de trabalhos com novos *frameworks* e tecnologias, foi dedicado um intervalo para que o desenvolvedor estudasse sobre o assunto abordado.

Destaca-se que cada projeto faz uso de um determinado *framework* e que a versão do Node.js utilizada em um determinado projeto pode não ser compatível com a estrutura e dependências de um outro sistema. Esse detalhe fez com que a versão do ambiente de execução utilizado no primeiro projeto fosse alterada para iniciar no segundo projeto.

3.4 Versionamento dos projetos

Em ambos os projetos era utilizado o sistema Git para realizar o versionamento dos projetos. Junto com o Git eram utilizadas as plataformas BitBucket e GitHub para alocar os repositórios dos projetos e cadastrar requisições de modificações.

Cada requisição para modificar arquivos nos repositórios deveriam ser aprovadas por, pelo menos, dois outros desenvolvedores. Caso esses revisores encontrassem alguma inconfor-

midade nas modificações, o desenvolvedor deveria ajustar os arquivos e pedir para o revisor verificar as novas alterações.

3.5 Projeto A

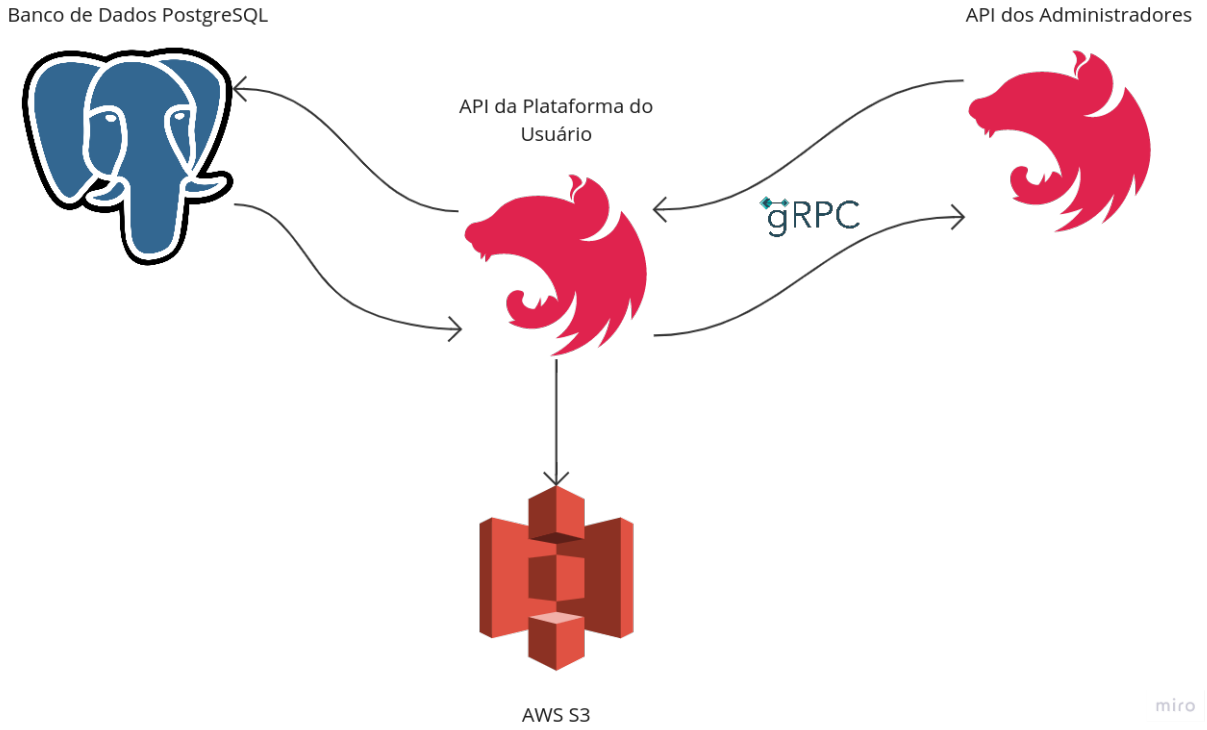
O projeto A corresponde a um sistema interno para gerenciamento de um determinado evento da empresa. Esse projeto seria responsável por oferecer todas as etapas disponíveis dentro do evento, desde a parte de inscrição até a finalização das atividades. Para isso, o projeto inicialmente foi pensado apenas com a construção de uma página web em que os participantes poderiam se inscrever no evento e realizar as atividades propostas. Nesse projeto, o estagiário foi responsável, junto a outro desenvolvedor back-end, por iniciar todo o desenvolvimento da API, modelagem do banco de dados e a definição de documentação. O desenvolvimento desse projeto foi iniciado em agosto e finalizado em novembro de 2021.

Depois do início do desenvolvimento, houve a necessidade de criar uma plataforma web para a administração, que controlaria as edições dos eventos, ou seja, um painel em que os organizadores do programa pudessem alterar as configurações visuais da plataforma do usuário. Para um melhor aproveitamento das funcionalidades que já estavam desenvolvidas e que atendesse o novo módulo que seria incrementado ao sistema, foi definido que o primeiro sistema ficaria com todo o gerenciamento e comunicação com o banco de dados e o segundo sistema comunicaria com as funções implementadas pelo primeiro através da estrutura gRPC. Dessa forma, o serviço principal centralizava as informações e era acionado por um outro serviço quando precisava das informações manipuladas pela API principal.

Como apresentado na Figura 3.2, as duas APIs foram desenvolvidas utilizando o *framework* NestJS, o banco de dados relacional era um PostgreSQL e todos os arquivos de imagem ou vídeo utilizados nas plataformas, eram alocados em um AWS S3.

A partir da definição das tecnologias que seriam utilizadas, foram definidas duas tarefas principais: a modelagem do banco de dados e a criação da estrutura de pastas do projeto, seguindo a metodologia interna da empresa.

Figura 3.2 – Estrutura que simula aplicação do projeto A



Fonte: Autor

3.5.1 Comunicação da equipe

Durante todo o período do projeto, a comunicação foi constante com o líder técnico. O apoio sempre era fornecido pela liderança em caso de dúvidas e problemas mais complexos para os desenvolvedores do projeto A, que eram iniciantes na carreira. Além do líder técnico, outros funcionários da empresa sempre ajudaram nos desafios da implementação do projeto.

Foram utilizadas metodologias ágeis no decorrer do desenvolvimento do projeto, especificamente os eventos do Scrum e o quadro Kanban, para organizar as tarefas de acordo com o seu andamento. As reuniões diárias eram de extrema importância para atualizar o time sobre alguma dúvida das tarefas e principalmente sobre o andamento da implementação feita pelos desenvolvedores front-end, que dependiam diretamente do que estava sendo feito pelo estagiário. Dessa forma, era possível entender se a velocidade das entregas feitas pelo estagiário estavam de acordo com o esperado, já que se tratava da primeira experiência com desenvolvimento.

Como o Scrum é um *framework* feito para se adaptar com o time, pode ser aplicado com alterações de acordo com a necessidade. Nesse caso os principais eventos do Scrum foram seguidos durante a atuação do usuário no projeto, sem nenhuma alteração.

3.5.2 Modelagem do banco de dados

Apesar de o estagiário ficar responsável pela criação de estrutura de pastas do projeto, ele também contribui para o desenho do modelo do banco de dados relacional. Foram desenhadas as tabelas do banco de dados relacional de acordo com a primeira demanda, que era focada no participante do evento. Dessa maneira, não existiam funções para usuário, ou seja, todos usuários eram do tipo "participante" e tinham o mesmo acesso dentro da plataforma.

Além dos usuários, a plataforma deveria gerenciar as edições, as trilhas, as aulas e os possíveis arquivos disponibilizados durante as apresentações, como algum trecho de código de programação, um vídeo, um arquivo PDF, entre outros. Em muitos casos, os arquivos, diferentes de dados de texto e números, não são alocados dentro de banco de dados, nesses casos são utilizadas ferramentas como o AWS S3.

3.5.3 Desenvolvimento das rotas

As primeiras rotas desenvolvidas foram destinadas à plataforma dos participantes dos eventos oferecidos pela plataforma, iniciando pelos *endpoints* com relação aos usuários e à autenticação. A autenticação garante ao usuário que iniciou uma sessão, acesso aos outros recursos oferecidos pela API. As rotas relacionadas ao usuário e autenticação criadas foram:

- **Criar usuário**: rota utilizada para cadastrar os dados de um novo usuário na plataforma, ou seja, um participante inédito em todos os eventos da organização. O usuário deveria cadastrar os dados pessoais e uma foto, caso o participante já estivesse na base de dados, a requisição era rejeitada. Após o cadastro dos dados no banco, a API enviava um e-mail com um código para o endereço cadastrado, para que fosse usada como confirmação da autenticidade de quem registrou os dados.

- **Confirmar criação de conta**: esse *endpoint* era responsável por receber o código enviado para a caixa de e-mails do usuário cadastrado para confirmar a criação da conta. Se o código JWT estivesse expirado, a requisição seria rejeitada pela API.

- **Atualizar usuário**: essa rota foi desenvolvida para fornecer ao usuário a possibilidade de atualizar os dados inseridos no cadastro, alguns dados não poderiam ser modificados, como documento (CPF).

- **Iniciar sessão de um usuário**: a rota responsável pela autenticação do usuário. Recebendo uma requisição com o e-mail e senha corretos do participante, esse *endpoint* adicionava um *cookie*[5] de autenticação na resposta da requisição. Se esses dados da requisição não estivessem corretos, era retornada uma mensagem de erro.

- **Reenviar e-mail para confirmar conta**: tinha a função de reenviar o e-mail com um novo código para que a conta criada pelo usuário pudesse ser confirmada na base de dados.

- **Enviar e-mail de esquecimento da senha**: esse *endpoint* era responsável por enviar um código através de um e-mail para o usuário, esse código poderia ser utilizado para trocar a senha.

- **Redefinir a senha**: utilizava o código, recebido pelo usuário no e-mail, e uma senha para definir uma nova senha para o participante.

- **Retornar dados do usuário conectado**: essa rota era responsável por retornar a maioria dos dados do usuário que estava logado. O participante logado era identificado por um *token* JWT que era recebido na requisição através de um *cookie*.

Apenas os *endpoints* responsáveis por atualizar o usuário e por retornar os dados do usuário logado, necessitavam de autenticação para serem acionados.

As outras atividades realizadas durante o desenvolvimento da API também eram no desenvolvimento de rotas das outras áreas englobadas pelo projeto. Como as rotas referentes aos eventos desenvolvidos, que contava com *endpoints* de criar os eventos e listar os eventos cadastrados na base de dados. Esses eventos contavam com atividades dentro da plataforma e essas tarefas deveriam ser separadas por seus temas, sendo necessário o desenvolvimento das rotas de criação, listagem e atualização das próprias entidades e das áreas que pertencem, durante o período de atuação do estagiário no projeto. Diferente das rotas referentes aos usuários, todas as rotas ligadas aos eventos demandava, ao usuário, estar autenticado para que as operações fossem realizadas.

[5] Cookies são pequenos arquivos de textos criados por aplicações e salvos no navegador para diversas finalidades, entre elas a autenticação de um usuário.

3.5.4 Área do administrador

Com o acréscimo das funcionalidades do serviço utilizado pela administração dos eventos, mudanças na base de dados e a configuração da comunicação gRPC foram devidamente realizadas na aplicação. Os endpoints da API do administrador acessava diretamente as funções para manipular o banco de dados, sendo toda a validação e regra de negócio feita na interface original, ou seja, a API utilizada pela página do administrador.

Essa aplicação contava com as rotas para usuários específicos, criando a demanda de separar os tipos de usuários por funções. Para criar as funções de cada usuário, foi necessário realizar adaptações no banco de dados.

3.6 Projeto B

O segundo projeto que o estagiário participou foi de uma empresa de tintas. A atuação nesse projeto se iniciou no final do estágio, em dezembro de 2021, e se prolongou durante um período, em que o estagiário já havia se efetivado dentro da empresa. Dentro desse projeto existiam diversas frentes, sendo aplicações que utilizavam APIs para fornecerem dados para os diversos módulos existentes dentro do projeto. Cada um desses módulos contavam com um microsserviço que acessava diretamente as outras APIs, que gerenciavam e formatavam os dados para atender as necessidades da aplicação que se referia, caracterizando esse microsserviço como um BFF (Back-end for Front-end). Como os dois projetos utilizavam tecnologias diferentes e o projeto B era de grande complexidade, ocorreram dificuldades inicias para o entendimento da organização e a comunicação dos diferentes módulos do projeto.

O módulo trabalhado pelo estagiário é referente a uma aplicação para dispositivos móveis que fornece informações sobre as cores disponibilizadas pela empresa, podendo ser feitas simulações e também formar coleções utilizando essas cores. Quando iniciou no projeto, o trabalho já havia se iniciado e algumas funcionalidades estavam implementadas, como a autenticação dos usuários e a listagem das cores. Outras funcionalidades foram desenvolvidas depois da entrada do estagiário no projeto.

3.6.1 Atividades e Funcionalidades

Como citado anteriormente, quando o estagiário foi alocado o projeto já estava em andamento. Dessa forma, antes de iniciar as atividades necessárias foi preciso conhecer o contexto do projeto e as tecnologias utilizadas para a implementação. Nesse projeto, o estagiário contou com a ajuda de um desenvolvedor que estava saindo desse módulo e dos desenvolvedores responsáveis por outros módulos, para entender sobre o projeto e as tecnologias. Depois do período de adaptação, o estagiário voltou a utilizar o serviço AWS Lambda.

As primeiras funcionalidades construídas, com a participação do estagiário, foram as rotas referentes as combinações de cores: o CRUD da entidade de combinações, *endpoints* para alocar cores em combinações e as operações para que o usuário conseguisse favoritar determinada combinação. Anteriormente à implementação dessas rotas, o desenvolvedor trabalhou na resolução de problemas reportados pelo time de front-end e pela Scrum Master. Na figura 3.3 são mostradas duas telas do sistema com as funcionalidades em que o estagiário mais trabalhou, relacionadas com cores e combinações.

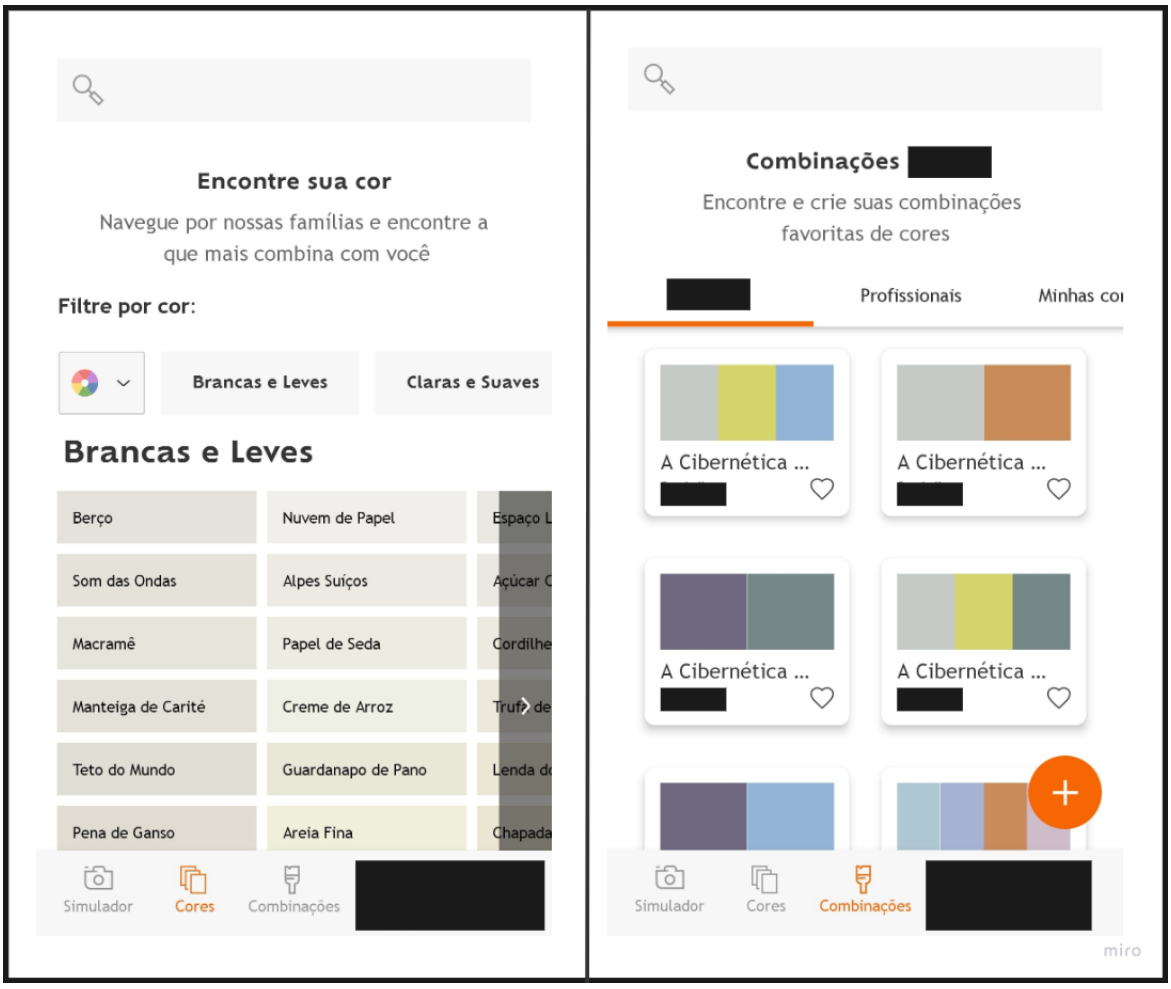
Depois de construir as funcionalidades diretamente relacionadas a combinações, surgiu a necessidade de fornecer ao usuário final a possibilidade de separar essas combinações em pastas para facilitar a manipulação. Dessa forma, o estagiário trabalhou no desenvolvimento de rotas referentes ao CRUD de pastas de combinações e o endpoint com a funcionalidade de associar uma combinação para pasta.

Em algumas situações, os desenvolvedores back-end contribuíam em módulos que não estavam alocados, isso ocorria em casos de funcionalidades ou correções grandes que podiam gerar impactos nos outros módulos. Esse trabalho em diferentes módulos, fez com que o estagiário que trabalhava no módulo referente ao aplicativo móvel entendesse como funcionava outras partes do projeto B.

3.6.2 Comunicação da equipe

Cada módulo contava com uma equipe específica e um Scrum Master que realizava os ritos em todos os módulos. Essas equipes seguiam as diretrizes do Scrum e todos os eventos especificados pelo *framework*. Reuniões semanais também ocorriam entre os desenvolvedores back-end, para alinhar alguma modificação feita nos serviços e para que demonstrações fossem

Figura 3.3 – Telas da aplicação *mobile* do projeto B



Fonte: Autor

realizadas para o restante da equipe, como aplicação de alguma nova tecnologia ou melhorias nos sistemas. No projeto B, também era utilizado o quadro Kanban para organizar as tarefas das aplicações.

4 CONCLUSÃO E CONSIDERAÇÕES FINAIS

Como relatado neste trabalho, a primeira experiência profissional é rica em desafios para o desenvolvedor. Por outro lado, contribui muito para adquirir conhecimento e também para aplicar os conhecimentos obtidos durante a formação acadêmica. Os desafios apresentados durante o período profissional foram superados pelos aprendizados recebidos, durante o curso de graduação, e também pela forma que a empresa recebe os novos colaboradores.

Grande parte do sucesso do período relatado está relacionado com as disciplinas ofertadas durante a graduação. Todas as disciplinas têm sua parcela de importância, apesar disso, algumas são mais importantes para a área do estágio relatado neste documento. As disciplinas que se relacionam diretamente com desenvolvimento e entendimento das estruturas de linguagens de programação, como “Introdução aos Algoritmos” e “Estruturas de Dados” que são base para atender o desenvolvimento. Outras disciplinas ofertadas na graduação, tem relação com o planejamento dos projetos, com o levantamento dos requisitos, metodologias ágeis e normas utilizadas durante a gestão e planejamento de projetos de software, uma dessas disciplinas é a “Engenharia de Software”. As disciplinas relacionadas com o tema “Banco de Dados”, como “Introdução a Sistemas de Banco de Dados” e “Sistemas Gerenciadores de Banco de Dados”, também foram de extrema valia durante o período relatado, considerando que no momento do estágio os desenvolvedores back-end eram responsáveis por esse gerenciamento dentro da organização.

Seria importante o curso de graduação ofertar disciplinas, na modalidade obrigatória, que abordassem práticas do desenvolvimento *Web* e *Mobile* utilizando as linguagens de programação e tecnologias mais utilizadas atualmente no mercado profissional. Dessa forma, o aluno conseguiria chegar mais preparado para aplicação dessas modalidades de desenvolvimento, muito utilizadas no ambiente profissional atualmente.

O programa "Camp"foi de extrema importância para preparar o estagiário para o ambiente que seria encarado, tendo em vista que a segunda fase do programa simulou, com algumas modificações, o funcionamento e a metodologia aplicada em um projeto da organização. Além disso, o programa ofertou a oportunidade da aplicação prática de metodologias ágeis aprendidas durante a graduação. O período de adaptação com um mentor técnico fornecido pela empresa, também transferiu conhecimentos sobre as ferramentas utilizadas atualmente na área de desenvolvimento e como aplicá-las na prática.

A empresa ofertou para o estagiário todo suporte profissional e pessoal, o que fortaleceu o vínculo e reduziu a curva de aprendizagem do aluno. É importante destacar que a vivência em um projeto interno da empresa facilitou a primeira experiência do estagiário com uma situação real, diminuindo a pressão criada em um novo ambiente de trabalho.

Por fim, conclui-se que programas de estágio em organizações que oferecem oportunidade de utilização de diferentes ferramentas e suporte para os colaboradores, são de extrema importância para o crescimento profissional e para o aprendizado dos alunos, aproximando a teoria aprendida durante o período da graduação e a prática ofertada pela empresa. Com isso, a oportunidade ofertada por empresas, através da formação adquirida na Universidade Federal de Lavras, contribui de forma significativa para a carreira de um possível desenvolvedor back-end.

REFERÊNCIAS

AUTH0. **Introduction to JSON Web Tokens**. 2023. Disponível em: <https://jwt.io/introduction>. Acesso em: 12 mar. 2023.

AWS. **O que é API RESTful?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/restful-api/>. Acesso em: 5 fev. 2023.

AWS. **O que é uma API?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/api/>. Acesso em: 5 fev. 2023.

BECK, K. et al. **Agile Manifesto**. 2001. Disponível em: <http://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 8 jan. 2023.

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

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

FONSECA, E. **O que é ORM?** 2019. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-orm>. Acesso em: 12 mar. 2023.

GRPC. **Introduction to gRPC**. 2022. Disponível em: <https://grpc.io/docs/what-is-grpc/introduction/>. Acesso em: 24 jan. 2023.

LAUBE, K. P. **Swagger na prática**. 2018. Disponível em: <https://klauslaube.com.br/2018/03/15/swagger-na-pratica.html>. Acesso em: 12 mar. 2023.

LECHETA, R. R. **AWS para Desenvolvedores**: Aprenda a instalar aplicações na nuvem da amazon aws. [S.l.]: Novatec Editora, 2014.

MONTANHEIRO, L. S.; CARVALHO, A. M. M.; RODRIGUES, J. A. Utilização de json web token na autenticação de usuários em apis rest. **XIII Encontro Anual de Computação. Anais do XIII Encontro Anual de Computação EnAComp**, p. 186–193, 2017.

NESTJS. **Hello, nest!** 2017. Disponível em: <https://nestjs.com/>. Acesso em: 6 fev. 2023.

NETO, W. **Construindo APIs testáveis com Node.js**. 2020. Disponível em: <https://leanpub.com/construindo-apis-testaveis-com-nodejs>. Acesso em: 24 jan. 2023.

NPM. **npm Docs**. 2023. Disponível em: <https://docs.npmjs.com/>. Acesso em: 5 fev. 2023.

OLIVEIRA, J. V. M. **Analisando a eficiência da comunicação entre serviços utilizando REST ou gRPC**. Universidade Federal do Rio Grande do Norte, 2021. Disponível em: <https://antigo.monografias.ufrn.br/jspui/bitstream/123456789/12486/1/AnalisandoEficienciaComunicacao_Oliveira_2021.pdf>. Acesso em: 13 jan. 2023.

PLÁCIDO, D. G.; CUNHA, D. P. da. Impedância em bancos de dados. **Anais SULCOMP**, v. 4, 2008.

POSTGRESQL. **Appendix D. SQL Conformance**. 2022. Disponível em: <https://www.postgresql.org/docs/current/features.html>. Acesso em: 24 jan. 2023.

SABINO, R. **Kanban: o que é, o Método Kanban, principais conceitos e como funciona no dia a dia**. 2023. Disponível em: <https://www.alura.com.br/artigos/metodo-kanban>. Acesso em: 12 mar. 2023.

SCHWABER, K.; SUTHERLAND, J. **O Guia do Scrum**: O guia definitivo para o scrum: As regras do jogo. 2020. Disponível em: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-3.0.pdf>. Acesso em: 12 jan. 2023.

SMARTBEAR. **OpenAPI specification**. 2020. Disponível em: <https://swagger.io/specification/>. Acesso em: 28 jan. 2023.

SOUZA, F. M. C.; LIMA, E. C. S.; CARIDADE, E. R. de S. Criando sistema escalável de agendamentos utilizando typescript com nestjs no backend e nextjs no frontend. **Revista Ibero-Americana de Humanidades, Ciências e Educação**, v. 8, n. 12, p. 43–57, 2022.

TYPEORM. **Migrations**. 2023. Disponível em: <https://typeorm.io/migrations>. Acesso em: 5 fev. 2023.

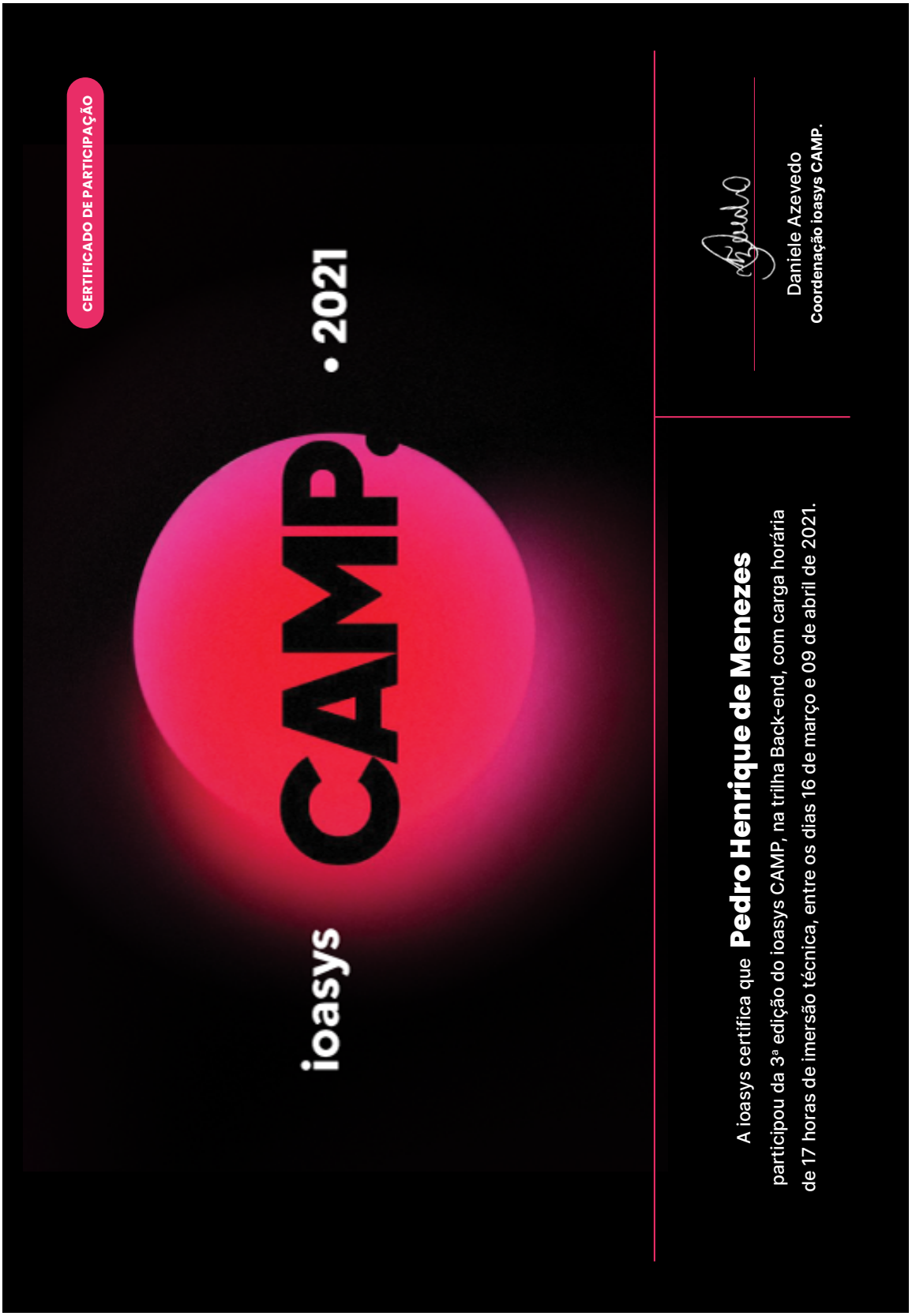
TYPESCRIPT. **TypeScript Documentation**. 2022. Disponível em: <https://www.typescriptlang.org/pt/docs/>. Acesso em: 5 fev. 2023.

VACANTI, D.; YERET, Y. **O Guia Kanban para Scrum Teams**. Scrum.org, 2021. Disponível em: <https://scrumorg-website-prod.s3.amazonaws.com/drupal/2021-02/2021-Kanban-Guide-Portuguese-Brazilian-2.0.pdf>. Acesso em: 13 jan. 2023.

VERAS, M. **Arquitetura de Nuvem (AWS): Amazon Web Services**. [S.l.]: Brasport, 2013.

VITALINO, J. F. N.; CASTRO, M. A. N. **Descomplicando o Docker 2a edição**. [S.l.]: Brasport, 2018.

ANEXO A – Certificados do programa Camp



FASE 2

CERTIFICADO DE PARTICIPAÇÃO

A ioasys certifica que **Pedro Henrique de Menezes**
participou da Fase 2 da 3ª edição do ioasys CAMP
entre os dias 11 de maio e 04 de junho de 2021.

Daniele Azevedo
Coordenação ioasys CAMP.