RODRIGO GUIMARÃES MARAFELLI PEREIRA

IMPLEMENTAÇÃO E MANUTENÇÃO DE APIS PARA O DESENVOLVIMENTO E EVOLUÇÃO DE SISTEMAS

LAVRAS - MG

2020

RODRIGO GUIMARÃES MARAFELLI PEREIRA

IMPLEMENTAÇÃO E MANUTENÇÃO DE APIS PARA O DESENVOLVIMENTO E EVOLUÇÃO DE SISTEMAS

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

Prof. Dr. Ramon Gomes Costa
Orientador

LAVRAS - MG
2020

RODRIGO GUIMARÃES MARAFELLI PEREIRA

IMPLEMENTAÇÃO E MANUTENÇÃO DE APIS PARA O DESENVOLVIMENTO E EVOLUÇÃO DE SISTEMAS

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

Prof. DSc. Ramon Gomes Costa	DCC/UFLA
Prof. DSc. Renata Teles Moreira	DCC/UFLA
Otávio Marques Anselmo	ioasys

Prof. DSc. Ramon Gomes Costa
Orientador

LAVRAS - MG
2019

Dedico este trabalho aos meus pais Ricardo Rabelo Guimarães e Sônia Maria Marafelli Pereira Guimarães, à minha namorada, Laura Rodrigues Gonçalves, à minha família e a todos os meus amigos.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais e namorada, por me darem todo apoio necessário para enfrentar e vencer todos os obstáculos que encontrei durante a graduação, eles foram os principais contribuintes no objetivo de concluir a graduação preservando a minha sanidade mental. Agradeço também a todos os meus amigos que, ao meu lado, foram imprescindíveis no caminho e conclusão desta jornada tão difícil. Por fim, gostaria de agradecer ao meu orientador Prof. Dr. Ramon Gomes Costa, por todos os ensinamentos na disciplina de Sistemas Gerenciadores de Bancos de Dados e pela orientação na escrita deste documento.
A todos vocês meu muito obrigado.

RESUMO

Com a evolução da tecnologia e consequentemente, dos meios de comunicação, há, cada vez mais, a necessidade da utilização de sistemas distribuídos para que a troca de informações seja feita de forma eficaz. Neste trabalho, são apresentadas as atividades realizadas durante o estágio na ioasys (Innovation Oasys Ltda.). Com atividades de administração de banco de dados e de utilização do protocolo de comunicação HTTP em projetos sob demanda com a utilização de metodologias ágeis para o desenvolvimento de novas funcionalidades e a resolução de problemas relatados pelos clientes, foram feitas implementações e manutenções de endpoints pertencentes à APIs, a fim de manter os dados seguros, consistentes e de fácil acesso a qualquer sistema que precisasse utilizá-los, como aplicativos, *sites*, e até mesmo outras APIs. A falta de padrões de desenvolvimento de código e a dificuldade em gerenciar o tempo para atender todas as demandas dos projetos foram considerados como pontos a serem melhorados.

Palavras-chave: API; banco de dados; metodologia ágil; protocolo; HTTP;

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Dentre momentos importantes na vida de um estudante, a graduação torna-se um momento único. Em frente a dificuldades e conquistas proporcionadas pela mesma, é necessário salientar a importância do complemento às atividades acadêmicas.

Sabe-se que, durante a graduação, desenvolve-se conhecimentos muito importantes, tanto na formação acadêmica, como na profissional. Com o objetivo de complementar as experiências adquiridas no curso de graduação em Ciência da Computação cursado na Universidade Federal de Lavras, foram exercidas atividades de estágio na Innovation Oasys Desenvolvimento de Sistemas Ltda (ioasys), que desenvolve softwares sob demanda e apoia empresas no processo de transformação de negócios, entregando experiências, as ajudando a se adaptarem ao cenário de transformação digital através de tecnologia, metodologias ágeis e estratégia.

A ioasys foi fundada em Belo Horizonte - MG, no ano de 2012. Seu principal objetivo é fornecer soluções digitais a outras empresas. Possui sua sede principal em Belo Horizonte, que conta com mais de 100 colaboradores, e outras sedes em cidades como Lavras - MG, Aracaju - SE, São Paulo - SP e Londres, totalizando mais de 150 colaboradores. Sua ideia é apoiar empresas, ajudando-as a se adaptarem ao novo cenário de transformação digital através de tecnologia, metodologias ágeis e estratégia.

As principais área de atuação da ioasys são: Desenvolvimento mobile; marketing, aplicativos sob demanda; *mobile commerce*, mobile *design*, *mobile enterprise*, *quality assurance*, *social apps*, *prototyping*, UX UI e transformação digital.

A ioasys possui como principal produto o Noz - Plataforma de Comunicação, um aplicativo de comunicação interna que ajuda diversas empresas a simplificar sua comunicação interna, consequentemente a tornando mais eficaz. O sistema possibilita um *layout* personalizado para cada empresa e possui um viés de "rede social", que possibilita que os usuários interajam através de publicações, enquetes e cursos. Além disso, também possui uma moeda interna, que, adquirida através de interações que o usuário tem com a plataforma, desde curtir uma publicação até concluir um curso, serve para aquisição de produtos da loja, pensada e mantida pela administração da empresa. Além de ser usado internamente pela própria ioasys, o Noz é distribuído e usado por outros clientes.

Este trabalho tem como objetivo apresentar as atividades realizadas no período de 1 ano de estágio, no período de vigência de 26/11/2018 a 26/11/2019, com carga horária de 30 horas semanais, totalizando 1440 horas, dentro da empresa ioasys. Pretende-se detalhar metodologias e ferramentas utilizadas tanto para desenvolvimento, como para armazenamento persistente de dados. Para isso, o trabalho está dividido em: Contextualização do mercado e apresentação da empresa; Tecnologias Utilizadas; Atividades desenvolvidas; Implementação de APIs para a evolução e manutenção de sistemas; e Considerações finais.

2 CONTEXTUALIZAÇÃO DO MERCADO E APRESENTAÇÃO DA EMPRESA

Este capítulo fornece uma descrição geral da empresa onde o estágio foi realizado, incluindo informações sobre o histórico, descrição física, plataforma de produtos, mercados atendidos pela empresa, número de funcionários, entre outros.

2.1 ioasys

A ioasys foi fundada em 2012, em Belo Horizonte - Minas Gerais, com foco em transformação digital, tecnologia e desenvolvimento. Em fase inicial, começou com poucos colaboradores (cerca de 5), atendendo poucos clientes. Em 2017, a ioasys já era responsável por todo o setor *mobile* do banco Inter, que é um banco digital mineiro, e pelo Viajanet, concorrente direto da Decolar.com. A ioasys também é uma das responsáveis e produtora oficial do Startup Games, um jogo de investimento estratégico entre *startups*. Hoje, com mais de 150 colaboradores, atende clientes não só no Brasil, mas também do exterior. A ioasys conduz diversos projetos em conjunto com empresas como Banco Inter, Burger King, BASF, VR Benefícios, LATAM Airlines, Pfizer, Localiza, Fleury, DMCard, Fundação Dom Cabral e Suvinil. Em 2017, a ioasys teve seu trabalho reconhecido pelo governo britânico, sendo contemplada pela British Brazilian Awards, prêmio de reconhecimento a empresas que estão se destacando no mercado britânico. O evento contou com mais de 250 dos maiores líderes corporativo dos dois países.

2.2 Mercado

As principais áreas de atuação da ioasys são: Desenvolvimento Mobile, Desenvolvimento iOS, Desenvolvimento Android, Marketing Mobile, Aplicativos Corporativos, Desenvolvimento Nativo, Aplicativos sob demanda, *Mobile Commerce*, *Mobile Design*, *Mobile Enterprise*, *Quality Assurance*, *Social apps*, *Prototyping*, UX UI, Transformação Digital.

A ioasys visa criar soluções digitais para empresas, utilizando inovação, design e tecnologia para auxiliar clientes no alcance de soluções que transformem a sociedade.

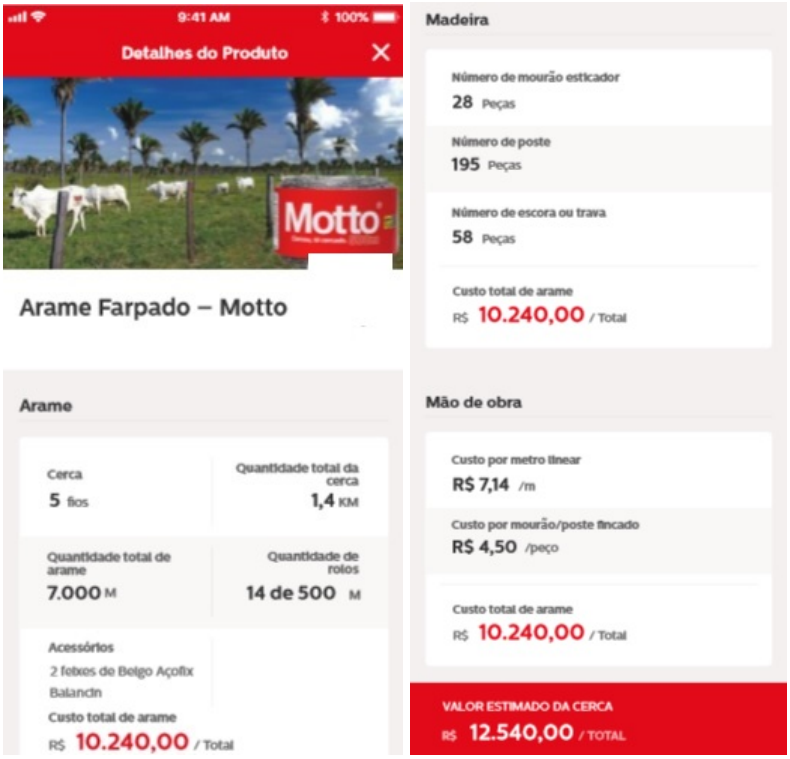
2.3 Participações em Projetos

Durante o estágio, depois de um treinamento para se adequar às tecnologias e ferramentas usadas pela empresa, o estagiário participou de quatro projetos, sendo alguns em fase de desenvolvimento, outros em fase de manutenção, e um que se trata de um produto interno da iaosys, que se encontra em contínua fase de desenvolvimento e manutenção. Sendo os projetos: Projeto A; Projeto B; Projeto C e Projeto D.

2.3.1 Projeto A

O Projeto A (Figura 2.1) foi desenvolvido para determinação de área para cercamento. O sistema possibilita que o usuário faça um orçamento de cerca para um determinado terreno. No aplicativo, o usuário faz a marcação da área utilizando o Google Maps e, após responder algumas perguntas sobre quais materiais deseja utilizar como matéria prima, o aplicativo faz os cálculos e exibe o orçamento.

Figura 2.1 – Projeto A - Orçamento de cerca



Quando o estagiário adentrou ao projeto, o mesmo se encontrava em fase final de desenvolvimento. Apesar disso, o estagiário conseguiu, atuando em conjunto com outro desenvolvedor experiente, contribuir com o projeto desenvolvendo algumas funcionalidades no escopo de administrador como: criação de usuários, listagem de lojas, autenticação de usuário administrador, entre outras.

2.3.2 Projeto B

O Projeto B (Figura 2.2) tem como principais objetivos permitir que atletas sejam avaliados de acordo com suas habilidades e encontrados por clubes e agentes. O aplicativo possibilita que tanto atletas como olheiros

se cadastrem. Do ponto de vista do atleta, é possível que ele visualize todos os eventos em sua região com data próxima, se inscreva e realize o pagamento da inscrição. Do ponto de vista do olheiro, é possível que o mesmo visualize os eventos disponíveis e os atletas inscritos nestes. Assim, com a realização do evento e através dos cálculos e análises feitos pelo aplicativo, o olheiro pode decidir-se sobre seu interesse perante os atletas.

Figura 2.2 – Projeto B - Listagem e detalhes de eventos

O estagiário atuou no Projeto B realizando manutenção contínua, pois o projeto já estava sendo utilizado pelo cliente e, frequentemente eram reportados problemas sobre o projeto. Como este projeto não possuía um escopo fixo, muitas vezes o cliente pedia novas funcionalidades a serem implementadas, portanto o estagiário também atuou desenvolvendo-as.

2.3.3 Projeto C

O Projeto C se trata de um aplicativo e sistema *web* que foi criado para uma empreiteira. A empresa possuía vários funcionários que atuavam tanto em escritórios como em campo. Portanto, havia necessidade de se fazer um controle dos empreendimentos sobre informações como: cadastro de anormalidades, atividades extra escopo, paralisações e outros artefatos que poderiam interferir no progresso de uma obra. O aplicativo também funcionava de forma *offline*.

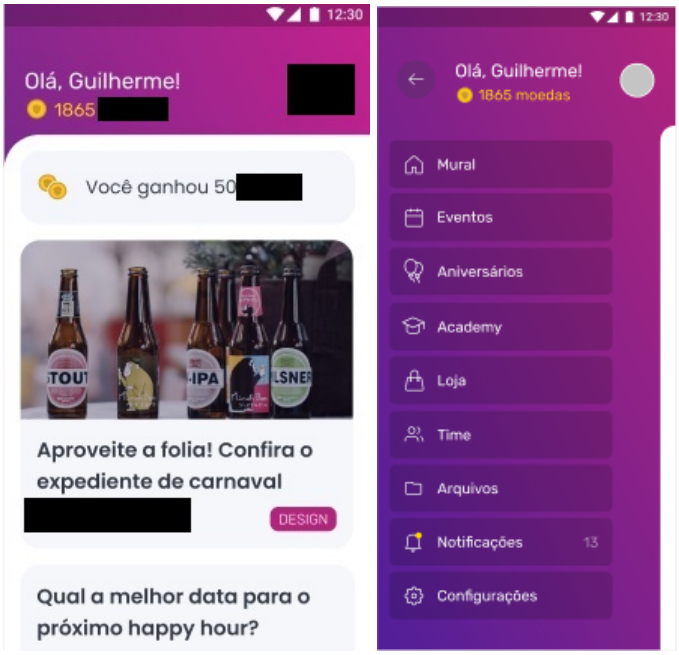
O estagiário atuou neste projeto desde a fase inicial até a fase final de desenvolvimento. Tendo como principais funcionalidades desenvolvidas a de sincronização de dados, que foi muito importante para o funcionamento *offline* do aplicativo e as notificações para o sistema *web*, que demandou estudo sobre o uso de *websockets*.

2.3.4 Projeto D

O Projeto D (Figura 2.3) é um sistema desenvolvido e mantido na forma de produto. O sistema é uma espécie de rede social interna que pode ser utilizada por qualquer empresa. Nele é possível criar publicações, enquetes, adicionar cursos, disponibilizar e comprar produtos através de uma moeda interna, acumulada através de interações do usuário com o aplicativo, curtindo, comentando e visualizando publicações, por exemplo. Este sistema vem ganhando bastante visibilidade no mercado, sendo utilizados por diversas empresas.

O estagiário atuou neste projeto durante vários meses. Como se tratava de um produto interno da empresa, o projeto possuia escopo aberto.

Figura 2.3 – Projeto D - Feed de Publicações e Menu



Por este motivo, o estagiário precisava não só corrigir problemas, como também desenvolver muitas novas funcionalidades.

3 TECNOLOGIAS UTILIZADAS

Neste capítulo são apresentadas as tecnologias utilizadas durante o período de estágio na empresa ioasys.

3.1 PostgreSQL e MySQL

Os SGBDS PostgreSQL [1] e MySQL [2] são Sistemas Gerenciadores de Banco de Dados (SGBD) objeto-relacional de código aberto.

Eles têm como características:

- "Comandos SQL são consistentes entre si e por padrão;
- É transacional, incluindo mudanças estruturais destrutivas;
- Suporta muitos tipos de dados sofisticados, incluindo *JSON*, *XML*, objetos geométricos, hierarquias, tags, e matrizes. Novos tipos de dados e funções podem ser escritos em *SQL*, *C*, ou linguagens procedurais muito incorporadas, incluindo *Python*, *Perl*, *TCL* e outras;
- Faz uso estratégico de indexação e consulta de otimização para trabalhar com o menor esforço possível" (CARVALHO, 2017).

O PostgreSQL é *Open Source* e gratuito. Já o MySQL possui versão gratuita e versão *enterprise*, que é paga.

Durante o desenvolvimento dos projetos A e C, ambos os SGBDs foram as ferramentas utilizadas para armazenar os dados da aplicação de forma persistente, mantendo a segurança e integridade dos dados. Além disso, por este foram realizadas consultas, criadas novas entidades, funções

[1] https://www.postgresql.org/
[2] https://www.mysql.com/

e atualizações de esquemas) através de *Object-relational mapping* (ORM) e *migrations* disponibilizadas pelos *frameworks* utilizados para o desenvolvimento do *backend* (*Active Record - Rails*) e (*Lucid - AdonisJs*). Estas atividades são descritas em seções posteriores.

3.2 Migration

Migration[3] é um recurso que permite a evolução do esquema do banco de dados ao longo do tempo. Com ela, em vez de gravar as mudanças do esquema utilizando a linguagem SQL, é possível escrever códigos utilizando a própria linguagem de desenvolvimento, simplificando o processo.

3.3 Adaptação do Scrum

Scrum[4] é uma metodologia ágil para gestão e planejamento de projetos de software (SCRUM, 2019). No *Scrum*, os projetos são divididos em ciclos, normalmente mensais chamados de *Sprints*, que na ioasys tem prazo de 10 dias úteis. A *Sprint* possui um conjunto de atividades que serão desenvolvidas durante a mesma.

É importante salientar que como em muitas empresas, a ioasys tem suas particularidades e desafios, fazendo com que não fosse possível a utilização do Scrum em sua plenitude, mas uma adaptação do mesmo.

As funcionalidades que devem ser implementadas no projeto são direcionadas para uma lista denominada *Backlog*, que é onde consta todas as demandas do projeto ainda não desenvolvidas. No início de cada *Sprint*,

[3] https://guides.rubyonrails.org/active_*record_migrations.html*
[4] https://www.desenvolvimentoagil.com.br/scrum/

faz-se uma reunião de planejamento, chamada *Planning*, que tem como objetivo planejar as atividades a serem desenvolvidas, onde *Product Owner* prioriza os itens do *Backlog* e, em conjunto da equipe que envolve desenvolvedores e *Scrum Master*, seleciona as atividades que possivelmente serão implementadas durante a *Sprint*. As tarefas delegadas à uma *Sprint* são movidas do *Backlog* para a lista de tarefas da *Sprint*, denominada *Sprint Tasks*, que geralmente possui o número que representa a *Sprint* no nome. As atividades que não foram concluídas, serão movidas para a próxima *Sprint* com maior prioridade para serem desenvolvidas.

A cada dia de uma *Sprint*, a equipe faz uma breve reunião, que pode ser feita em qualquer horário do dia, chamada de *Daily* onde todos relatam quais atividades estão desenvolvendo, e quais irão desenvolver em sequência, tiram e esclarecem dúvidas. Desta forma, todos envolvidos no projeto ficam informadas sobre o que as outras pessoas estão trabalhando, melhorando a comunicação e o sincronismo no desenvolvimento do projeto.

Ao final de uma *print*, a equipe apresenta as funcionalidades implementadas em uma *Review*. Finalmente, faz-se uma *Retrospectiva*, onde todos da equipe pontuam pontos positivos e negativos, estudando possíveis melhorias dos pontos negativos para as próximas *Sprints*. Após isso a equipe segue a *Planning*, que é o planejamento da próxima *Sprint*. Completando o ciclo.

Durante o estágio, nas equipes em que o estagiário foi alocado, na maioria dos projetos se trabalhava usando *Sprints*. Somente no projeto D, houve dificuldade de se utilizar *Sprints*, pois tratava-se de um projeto que já estava sendo utilizado por vários clientes e, frequentemente, era necessário realizar manutenções ou lançar novas versões (atualizar o código do projeto no servidor), que demandavam tempo e esforço do ponto de vista

do *backend*, que foi a principal área de atuação do estagiário. Por estes motivos, era difícil estabelecer tarefas fixas para serem desenvolvidas durante uma *Sprint*, pois durante a mesma, surgiam muitas tarefas não previstas, o que atrasava o desenvolvimento do *backend*.

3.4 Kanban

Kanban é um sistema para ser usado em metodologia ágil, para controle de produção ou gestão de tarefas (ESPINHA, 2019). Este divide-se em três partes:

- **Cartão**: o cartão é o menor fragmento do *Kanban*. Tem como objetivo retratar uma tarefa ou ação a ser executada para que o resultado final seja atingido;
- **Colunas**: as colunas representam o *status*, que é a condição na qual o cartão se encontra. Um quadro *Kanban* geralmente tem três colunas: "**A Fazer/To do**", "**Em Execução/Doing**"e "**Feito/Done**". É possível que existam mais colunas (*status*) a medida que for de necessidade da equipe.
- **Quadro**: o quadro é organizado em cartões e colunas. Geralmente utiliza-se um quadro *Kanban* para cada projeto, que é utilizado por todos da equipe, mas em projetos com equipes maiores, pode ser necessário a utilização de quadros separados por equipes.

Durante o estágio, o *Kanban* foi utilizado em todos os projetos em que o estagiário atuou e foi importante na avaliação do desenvolvimento dos projetos, pois com ele foi possível visualizar o curso do trabalho e as

atividades de cada *Sprint* do projeto. Isto facilitou a análise das atividades, a evolução dos projetos e a detecção de problemas durante o desenvolvimento, possibilitando que estes fossem resolvidos de forma mais rápida, contribuindo para o desenvolvimento ágil.

3.5 Bitbucket

Bitbucket[5]é um serviço de hospedagem de projetos utilizando o Mercurial, que é um sistema distrubuído de controle de versões. Similar ao GitHub (que utiliza somente Git), o Bitbucket têm um serviço grátis e um comercial. Ele é escrito em Python. Além disso, o Bitbucket suporta repositórios usando o sistema de controle de versões Git.

O Bitbucket possui várias funções. Algumas são:

- Repositórios gratuitos, privados e ilimitados: Gratuito para equipes de até 5 membros e com valores proporcionais nos planos Standard;
- Integração com Jira e Trello: Possibilita manter seus projetos organizados criando ramificações do Bitbucket direto dos itens do Jira ou cartões do Trello;
- Entrega contínua integrada: Permite construir, testar e implementar com CI/CD integrado. Aproveitando configurações como código e *loops* rápidos de *feedback*;
- Colaboração de código: Possibilita compilação de *software* de qualidade com revisão de código. Promove análise do código de forma

[5] https://bitbucket.org/

mais eficiente com solicitações *pull*, com lista de verificação mesclada com aprovadores designados, possibilitando manter discussões direto no código fonte com comentários nas linhas;

- Pipelines: Contém implementações que permitem que você construa, teste e implemente com CI/CD integrado. Aproveitando configurações como código e *loops* rápidos de *feedback*; entre outras.

O Bitbucket foi utilizado durante todo o perído de estágio. Nele, ficaram armazenados todos os repositórios contendo código fonte dos projetos em que o estagiário atuou. Toda submissão de código feita pelo estagiário passava por revisão de os outros desenvolvedores da equipe de backend da ioasys, onde era necessária aprovação de outros 2 desenvolvedores para que o código pudesse ser submetido para a Branch principal de desenvolvimento.

3.6 RoR - Ruby on Rails

O Ruby on Rais[6] é um framework livre que se propõe a ser utilizado no desenvolvimento de *sites* orientados a banco de dados (*database-driven web sites*), tornando possível criar aplicações baseadas em estruturas pré-definidas. O Ruby on Rails é escrito na linguagem de programação Ruby. As aplicações criadas utilizando o *framework* Rails são desenvolvidas com base no padrão de arquitetura MVC (Model-View-Controller).

O RoR possui várias funções. Algumas são:

- Active Record: É o M no MVC,o modelo que é a camada do sistema responsável por representar dados e lógica de negócios. O Active

[6] https://rubyonrails.org/

Record facilita a criação e o uso de objetos de negócios cujos dados requerem armazenamento persistente em um banco de dados. O Active Record é uma descrição de um sistema de Mapeamento Relacional a Objetos;

- Action Pack: Compreende o Action View (geração de visualização de usuário, como HTML, XML, JavaScript, entre outros) e o Action Controller (controle de fluxo de negócio);

- Action Mailer: É um framework responsável pelo serviço de entrega e até mesmo de recebimento de e-mails. É relativamente pequeno e simples, porém poderoso e capaz de realizar diversas operações apenas com chamadas de entrega de correspondência;

- Active Support: É uma coleção de várias classes úteis e extensões de bibliotecas padrões, que foram considerados úteis para aplicações em Ruby on Rails;

Na ioasys, o Ruby on Rails é utilizado em muitos projetos, devido a sua curva de aprendizado ser pequena, e possibilitar o rápido desenvolvimento e manutenção de projetos. Durante o período de estágio, foi o framekwork mais utilizado para o desenvolvimento e manutenção de APIs pelo estagiário.

3.7 AWS - Amazon Web Services

A Amazon Web Services[7] é uma plataforma de serviços de computação em nuvem oferecida pela Amazon[8].

[7] https://aws.amazon.com/pt/
[8] https://www.amazon.com/

Dentre os serviços oferecidos pela AWS, os mais utilizados foram:

- Amazon Elastic Compute Cloud[9]: Serviço que provê capacidade computacional segura e redimensionável na núvem.
- S3[10]: Serviço de armazenamento de objetos em pequena ou grande escala na nuvem.
- Relational Database Service (RDS)[11]: Serviço que provê o uso de bancos de dados relacionais na nuvem.

3.8 Outras ferramentas

Além das ferramentas citadas anteriormente, é importante citar outras duas que foram muito utilizadas durante a experiência na ioasys. Sendo elas:

- Redis[12]: Repositório de estrutura de dados em memória, utilizado como banco de dados, *cache* e intermediário de mensagens. Durante a experiência do desenvolvedor, o redis foi muito utilizado no envio de notificações para dispositivos móveis.
- Nginx[13]: Ferramenta que cumpre várias funções como: servidor HTTP, *proxy* reverso e balanceador de carga. Durante a experiência do desenvolvedor, foi utilizado principalmente como servidor HTTP. O Nginx direcionava as requisições que chegavam no servidor para a aplicação (API), que então processava as requisições.

[9] https://aws.amazon.com/pt/ec2/
[10] https://aws.amazon.com/pt/s3
[11] https://aws.amazon.com/pt/rds/
[12] https://redis.io/
[13] https://www.nginx.com/

4 ATIVIDADES DESENVOLVIDAS

Nesse capítulo, são apresentadas algumas das atividades desenvolvidas durante o período de estágio na ioasys. Também é apresentada uma relação entre conceitos aprendidos em sala de aula e o que foi utilizado no estágio.

4.1 Configuração do Ambiente

Quando se deseja desenvolver um sistema Web, de forma rápida, simples e segura, muitas empresas escolhem para essa tarefa o Ruby on Rails. A explicação para isso é que o Ruby on Rails possui uma série de ferramentas que facilita o desenvolvimento, auxiliando o desenvolvedor nas escolhas importantes. Resultando em uma rápida entrega do projeto, somada a um sistema que atende às necessidades do usuário. No momento em que se decide desenvolver uma aplicação utilizando Ruby On Rails, existe uma série de configurações prévias que devem ser feitas no ambiente tornando isto possível. A primeira etapa é a instalação do Ruby no ambiente e, para isso, precisa-se pensar em qual versão do Ruby utilizar. Esta escolha depende de alguns fatores: Algumas vezes é necessário uma versão mais antiga do Ruby, por exemplo a versão 2.7.1, quando precisar desenvolver ou realizar manutenção em sistemas legado, que foi desenvolvido há mais tempo, onde foram utilizadas versões antigas do Ruby. Outras vezes, é interessante optar pela versão mais recente do Ruby, quando for desenvolver novas aplicações por exemplo. Desta forma, aproveita-se do que o Ruby oferece de melhor.

Após decidida a versão do Ruby a ser utilizada, existem várias maneiras de fazer a instalação, uma delas é utilizando o Ruby Version Mana-

ger (RVM), que provê uma simples instalação e gerenciamento de versões Ruby, o que possibilita simultaneamente ter várias versões do Ruby instaladas no mesmo ambiente a utilização de várias versões pelo desenvolvedor.

Concluída a etapa de instalação do Ruby, a próxima etapa é a instalação do Rails, onde, assim como o Ruby, a versão depende da necessidade do desenvolvedor. A instalação do Rails pode ser feita por meio do comando 'gem install rails -v <versão>'. As bibliotecas do Ruby são chamadas de Gems, então o comando 'gem install <nome da gem> instala uma gem.

Com Ruby e Rails instalados, o próximo passo é escolher e instalar um SGBD e seguir o processo de instalação padrão do mesmo no ambiente em que o desenvolvedor estiver utilizando.

4.2 Primeiras Atividades

No início do período de estágio, as primeiras atividades desenvolvidas pelo estagiário foram as descritas acima. Devido a simplicidade de execução das mesmas, o estagiário não precisou de auxílio para concluí-las.

Após a configuração do ambiente, nas primeiras semanas do período de estágio, o estagiário ficou sob supervisão de um desenvolvedor *backend* contratado da ioasys, que mostrou-se sempre disponível para discutir e sanar qualquer tipo de dúvida, desde assuntos técnicos, até informações sobre a cultura e ambientação do estagiário na empresa. Nessas primeiras semanas, as principais atividades do estagiário foram fazer cursos sobre Ruby on Rails, que abordavam sobre o desenvolvimento de APIs, que seria sua principal atividade dali em diante.

4.3 Arquitetar, desenvolver e manter APIs

Interfaces de programação de aplicações (APIs) são, conjuntos de comandos, funções, protocolos ou objetos que terceiros podem usar para desenvolver ou interagir com sistemas externos. Todo esse conteúdo é disponobilizado para o desenvolvedor de forma pública ou privada, através de *endpoints*. Um *Endpoint* é uma das funcionalidades de uma API, que é acessível por terceiros através de uma *URL*.

Na ioasys, a grande maioria dos projetos tem seu *backend* desenvolvido em forma de API. Elas são responsáveis por fazer autenticação de usuários, validações e armazenamento de dados utilizando SGBDs relacionais, como MySQL e PostgreSQL.

Durante o período de estágio, algumas das atividades mais comuns para o estágiario eram arquitetar, desenvolver e manter APIs.

Após as primeiras semanas, quando o estagiário já havia concluído os cursos sobre Ruby on Rails e desenvolvimento de APIs, ele teve o primeiro contato com uma API de verdade começando a atuar no projeto A, onde, em conjunto com outro desenvolvedor mais experiente, desenvolveu alguns *endpoints* para a API do projeto, tais como: Autenticação de administrador; cadastro de administrador; listagem de usuários utilizando o filtro nome; listagem de franquias; pesquisa de franquias; cadastro de franquias; cadastro de cidade e estado de franquia; editar produtos removendo características; editar produtos removendo aplicações; editar franquia; remover franquia; cadastrar franquia através de importação de planilha; adicionar lojas ao produto; pesquisar produtos utilizando o filtro cidade; cadastrar administrador; editar administrador; remover administrador; listar administradores; busca de lojas contendo um determinando produto; excluir

usuário; pesquisar ordem; alterar senha do administrador logado; busca de administrador por nome; excluir administrador; desvincular todas as lojas de um produto; entre outros. Para cada endpoint, foi definida uma tarefa, representada por um cartão no quadro *Kanban*, cujo funcionamento foi explicado em sessões anteriores.

4.3.1 Projeto A

A (Figura 4.1) mostra um arquivo de *controller*. Nele é possível ver os métodos disponíveis e, cada um deles está relacionado a um *endpoint* correspondente. Sendo eles:

- **index**: acessível através do *endpoint* /attachments e utilizando o método HTTP GET, esse método lista todos os registros da tabela anexos do banco de dados, de forma paginada.

- **create**: acessível através do *endpoint* /attachments e utilizando o método HTTP POST, esse método insere um registro na tabela anexos do banco de dados, utilizando os parâmetros provenientes da requisição HTTP, definidos no método *params-attachment*.

- **update**: acessível através do *endpoint* /attachments/:id e utilizando o método HTTP PUT, esse método altera um registro na tabela anexos do banco de dados, utilizando os parâmetros provenientes da requisição HTTP, definidos no método *params-attachment*.

- **destroy**: acessível através do *endpoint* /attachments:id e utilizando o método HTTP DELETE, esse método exclui um registro na tabela anexos do banco de dados.

- **métodos privados**: os métodos *set-attachment* e *params-attachment* são métodos privados acessíveis apenas a componentes internos da aplicação, como os métodos citados acima. O método *params-attachment* define os parâmetros aceitos para criar ou atualizar um registro, enquanto o método *set-attachment* é um método chamado antes dos métodos *update* e *destroy*, sendo responsável por buscar um registro antes de alterá-lo ou excluí-lo.

Figura 4.1 – Projeto A - Arquivo de *controller* de anexos

```ruby
class Api::V1::Admin::AttachmentsController < Api::V1::Admin::BaseController
    before_action :authenticate_api_v1_admin_admin_user!
    before_action :set_attachment,except:[:index,:create]

    def index
        @attachments = Attachment.all
        params[:page] ||= 1
        params[:per_page] ||= 10
        @pagy, @attachments = pagy(Attachment.filter(by_name: params[:name]).order(name: :asc),items: params[:per_page],page: params[:page])
    end

    def create
        @attachment = Attachment.new params_attachment
        unless @attachment.save
            render json:{errors: @attachment.errors.full_messages}, status: 422
        end
    end

    def update
        unless @attachment.update(params_attachment)
            render json:{errors: @attachment.errors.full_messages}, status: 422
        end
    end

    def destroy
        Attachment.destroy(params[:id])
        render json:{message:"Catálogo removido com sucesso."}, status: 200
    end

    private

    def params_attachment
        params.permit(:id,:name,:attachment_type,:publish_at,:filename,:photo)
    end

    def set_attachment
        @attachment = Attachment.find(params[:id])
    end


end
```

É importante salientar que, para um desenvolvedor *backend* na ioasys, desenvolver uma funcionalidade quer dizer desenvolver um *endpoint*. Como exemplo disso podemos imaginar a funcionalidade de cadastro de usuários de um sistema. Para disponobilizar essa funcionalidade, o desenvolvedor cria um *endpoint*, que, acessível através de uma URL, recebe no

corpo da requisição os dados do usuário a ser inserido, faz as devidas validações, de acordo com as regras de negócio do sistema e, por fim, insere o usuário no banco de dados.

Endpoints são a parte mais externa de uma API, onde é feita a comunicação entre cliente e servidor. Para construir um endpoint, é necessário tomar uma série de decisões importantes que podem impactar tanto no funcionamento do endpoint, como na integridade dos dados dos usuários e na performance da aplicação. Estas decisões, que não foram definidas como tarefas para o desenvolvedor, por se te tratarem de questões técnicas importantes apenas ao desenvolvimento, e não na gestão de projetos. Assim, sempre que o estagiário precisou desenvolver um *endpoint*, era necessário saber se a aplicação estava pronta para disponibilizá-lo. Para isso, era importante: conhecer o banco de dados da aplicação, saber quais os dados e seus tipos; conhecer os relacionamentos entre tabelas; ter ciência de como funcionavam as transações; entre outras coisas. Nesta etapa, o estagiário pode observar os conhecimentos aprendidos nas disciplinas de Introdução ao Sistema de Banco de Dados e Sistemas Gerenciadores de Banco de Dados, onde aprendeu a teoria sobre bancos de dados e, dessa forma pôde colocá-la em prática realizando criação e atualização de tabelas; realização de consultas; inserção, atualização e remoção de registros; junção de tabelas; entre outras atividades.

O projeto A se encontrava em desenvolvimento durante o período de estágio. Por este motivo, o estagiário passou mais tempo desenvolvimento novos *endpoints* do que realizando manutenções em *endpoints* existentes.

Após a entrega do projeto A, o estagiário possuía conhecimentos básicos sobre Ruby on Rails e desenvolvimento de APIs. Então o estagiário foi alocado em outro projeto, chamado B.

4.3.2 Projeto B

Durante o período de estágio, o Projeto B, diferente do Projeto A, já se encontrava em funcionamento. Então, além de desenvolver novos *endpoints*, o estagiário também ficou responsável por realizar manutenções na API do projeto, em *endpoints* já implementados por outros desenvolvedores anteriormente.

Neste projeto o desenvolvedor começou a atuar em conjunto com outro desenvolvedor responsável pelo *backend*, que nas primeiras semanas foi responsável por auxiliá-lo, explicando as regras de negócio do projeto, padrões de implementações e fluxos de funcionamento.

Após o perído de adaptação, o estagiário ficou responsável por todo o backend do projeto, evoluindo e mantendo a API sozinho. Nesta etapa, criou os seguintes *endpoints*: disponibilizar documento em formato pdf para download, contendo os dados dos atletas inscritos em um determinado evento; mostrar relatórios dos atletas avaliados através do aplicativo; entre outros. E realizou outras tarefas como: corrigir que fazia o relatório ficar em branco no corpo do email; entender o processo de exclusão de eventos para realizar possíveis alterações; alteração de domínio da API no servidor de homologação.

As (Figuras 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 e 4.8) mostram alguns métodos definidos no *controller* de pagamentos, sendo eles:

- **check-address**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método verifica se um atleta já possui um endereço cadastrado e, caso não possua, um endereço é cadastrado. Isso é feito porque no momento de realizar o pagamento da inscrição de um evento, a biblioteca utilizada no pro-

jeto exigia que pagante tivesse um endereço, tanto para pagamentos com boleto, como pagamentos em cartão de crédito.

- **check-nationality**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método verifica se o atleta possui uma nacionalidade cadastrada e, caso não possua, seus dados de nacionalidade são atualizados. Isso é feito porque, assim como o endereço, a nacionalidade é um dado obrigatório para efetuar um pagamento.

- **create**: acessível através do *endpoint* /payments e utilizando o método HTTP POST, esse método cria um registro na tabela de pagamentos do banco de dados. Além disso, o método é responsável por realizar um pagamento via boleto ou cartão de crédito.

- **create-international-payment**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método cria um pagamento utilizando o passaporte do atleta como documento de identificação. Isso ocorre porque como só no Brasil se utiliza CPF, a biblioteca utilizada no projeto obrigou o número do passaporte como documento de identificação no momento do pagamento internacional.

- **create-payment-brazilian-underage**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método cria um pagamento utilizando o documento de CPF do responsável do atleta, no caso de atletas menores de idade.

- **create-payment-brazilian-ofage**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este

método cria um pagamento utilizando o documento de CPF do próprio atleta, no caso de atletas maiores de idade.

- **set-participant**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método busca na tabela de participantes de eventos o participante que quer realizar o pagamento da inscrição de um determinado evento.

Figura 4.2 – Projeto B - Verificação de endereço

```
def check_address
  if @participant.athlete.address.nil?
    address = Address.create!(address_attributes)
    @participant.athlete.update!(address_id: address.id)
  else
    @participant.athlete.address.update!(address_attributes)
  end
end
```

Figura 4.3 – Projeto B - Verificação de nacionalidade

```
private      You, a year ago • Changing payment flow

def check_nationality
  if (@participant.athlete.nationality.nil? || @participant.athlete.nationality.blank?)
    unless athlete_attributes[:nationality]
      render json: {errors: ["É necessário fornecer a nacionalidade."]}, status: :bad_request
      return false
    else
      country = Country.find(athlete_attributes[:nationality])
      nationality = country.name
      @participant.athlete.update!(nationality: nationality)
    end
  end
end
```

A atuação do estagiário no B foi de grande importância porque, como consequência de assumir novas responsabilidades, como atualizar a API do servidor de homologação e tomar decisões importantes sem o auxílio de outros desenvolvedores, o estagiário pode observar conceitos importantes aprendidos nas disciplinas de Redes de Computadores e Sistemas

Figura 4.4 – Projeto B - Criação de pagamento

```ruby
class Api::V1::PaymentsController < Api::V1::BaseController
  before_action :authenticate_user!
  before_action :set_participant, only:[:create]

  def create
    check_nationality
    check_address
    if @participant.athlete.is_brazilian?
      if Payment.athlete_underage?(@participant.athlete)
        create_payment_brazilian_underage
      else
        create_payment_brazilian_ofage
      end
    else
      create_international_payment
    end
  end
```

Figura 4.5 – Projeto B - Criação de pagamento internacional

```ruby
def create_international_payment
  if (athlete_attributes[:document_type] && athlete_attributes[:document_number])
    unless athlete_attributes[:document_type] == 'passport'
      render json: {errors: ["É necessário fornecer o Passaporte."]}, status: :bad_request
      return false
    else
      @participant.athlete.update!(document_type: athlete_attributes[:document_type], document_number: athlete_attributes[:document_number])
      create_transaction(@participant,payment_method[:card_hash],payment_method[:installments],payment_method[:payment_method])
    end
  else
    if (@participant.athlete.document_type.present? && @participant.athlete.document_number.present?)
      create_transaction(@participant,payment_method[:card_hash],payment_method[:installments],payment_method[:payment_method])
    else
      render json: {errors: ["É necessário fornecer o Passaporte."]}, status: :bad_request
      return false
    end
  end
end
```

Figura 4.6 – Projeto B - Criação de pagamento nacional para menor de idade

```ruby
def create_payment_brazilian_underage
  if athlete_attributes[:responsable_cpf]
    @participant.athlete.update!(responsable_cpf: athlete_attributes[:responsable_cpf])
    create_transaction(@participant,payment_method[:card_hash],payment_method[:installments],payment_method[:payment_method])
  else
    unless @participant.athlete.responsable_cpf.nil?
      create_transaction(@participant,payment_method[:card_hash],payment_method[:installments],payment_method[:payment_method])
    else
      render json: {errors: ["É necessário fornecer o CPF do responsável."]}, status: :bad_request
      return false
    end
  end
end
```

Distribuídos como: funcionamento do protocolo HTTP, certificação SSL, entre outros. E desta forma, pode colocar estes conhecimentos em prática

Figura 4.7 – Projeto B - Criação de pagamento nacional para maior de idade

```
def create_payment_brazilian_ofage
  if (athlete_attributes[:document_type] && athlete_attributes[:document_number])
    unless athlete_attributes[:document_type] == 'cpf'
      render json: {errors: ["É necessário fornecer o CPF."]}, status: :bad_request
      return false
    else
      @participant.athlete.update!(document_type: athlete_attributes[:document_type], document_number: athlete_attributes[:document_number])
      create_transaction(@participant,payment_method[:card_hash],payment_method[:installments],payment_method[:payment_method])
    end
  else
    if (@participant.athlete.document_type.present? && @participant.athlete.document_number.present?)
      create_transaction(@participant,payment_method[:card_hash],payment_method[:installments],payment_method[:payment_method])
    else
      render json: {errors: ["É necessário fornecer o CPF."]}, status: :bad_request
      return false
    end
  end
end
```

Figura 4.8 – Projeto B - Buscar participante de evento para efetuar pagamento

```
def set_participant
  @participant = Participant.where(athlete_id: payment_method[:athlete_id], eventable_id: payment_method[:event_id]).first
end
```

realizando atividades como: conexão em servidores remotos utilizando o protocolo Secure Shell (SSH); geração de certificado Secure Sockers Laye (SSL);

4.3.3 Projeto C

No projeto C, que também estava em fase de desenvolvimento, o estagiário, em conjunto com outro desenvolvedor backend, ficou responsável por desenvolver novos endpoints para o projeto, visando uma entrega rápida. Neste projeto o estagiário criou os seguintes endpoints: cadastro de usuários com envio de email; editar atividade extra escopo; excluir imagens e vídeos de atividade extra escopo; excluir mão de obra direta; excluir equipamento; alterar senha do usuário; editar anormalidade; remover anormalidade; cadastro de emprendimento; possibilitar *upload* de arquivos de atividades extra escopo e de improdutividade; sincronização de dados para

o aplicativo, para possibilitar o funcionamento *offline* do mesmo, mantendo a integridade dos dados; entre outros.

As (Figuras **??**, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16 e 4.17) mostram alguns métodos definidos no Projeto C, sendo eles:

- **amountOfDaysWithoutRdo**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método é executado diariamente para identificar os registros diários de obras (RDOs) não assinados. Isso era necessário porque devido as regras de negócio do projeto, os RDOs deveriam ser registrados e assinados todos os dias no decorrer das obras de um empreedimento.

- **sign**: acessível através do *endpoint* /sign e utilizando o método HTTP POST, esté método é responsável por realizar a autenticação de um usuário no sistema.

- **signRdo**: acessível através do *endpoint* /rdos/:id/sign e utilizando o método HTTP POST, este método serve para que um RDO seja assinado.

- **createRdo**: acessível através do *endpoint* /rdos e utilizando o método HTTP POST, este método cria um registro na tabela de rdos no banco de dados.

- **checkEnterpriseRdos**: método acessível apenas a própria aplicação, nao sendo disponibilizado através de um *endpoint*. Este método é executado diariamente para identificar a quantidade de dias durante as obras do empreendimento em que os RDOs não foram assinados.

- **index**: acessível através do *endpoint* /rdos e utilizando o método HTTP GET, este método lista todos os registro da tabela de rdos no banco de dados, de forma paginada.

- **newDataSync**: acessível através do *endpoint* /synchronization e utilizando o método HTTP GET, este método retorna ao usuário que fiz a requisição a cópia atual dos registros de todas as tabelas do banco de dados. Desde os registros novos, atualizados e os excluídos. Este método foi necessário porque o aplicativo do projeto precisava funcionar quando o usuário não tivesse conexão com a internet. Assim, sempre que possível, uma cópia do banco de dados era mantida no próprio aplicativo.

Figura 4.9 – Projeto C - Autenticação

```
async sign({ request, params, response }) {
  const id = params.id
  try {
    if (id) {
      await Service.signRdo(id, request)
      return response
        .status(200)
        .json({ message: 'RDO assinado com sucesso!' })
    }
    return response.status(404).json({ errors: ['RDO não encontrado!'] })
  } catch (error) {
    console.error(error)
    if (error.errors) {
      return response.status(error.status).json({ errors: [error.errors] })
    }
    return response.status(406).json({ errors: [error] })
  }
}
```

Figura 4.10 – Projeto C - Assinatura de RDO

```
static async signRdo(id, request) {      You, a year ago • creating rdoId
  const rdo = await Rdo.find(id)

  if (!rdo) throw `RDO não encontrado no sistema`
  if (rdo.toJSON().status === 'signed') throw `RDO já assinado no sistema`

  const rdoNumber = rdo.toJSON().rdoNumber.replace(/\//g, '-')
  let filePath = `rdos/${id}/${rdoNumber}`

  try {
    request.multipart.file('pdf', {}, async file => {
      filePath = `${filePath}.${file.extname}`
      await Drive.disk('s3').put(filePath, file.stream)
    })

    await request.multipart.process()

    const fileUrl = await Drive.disk('s3').getUrl(filePath)

    rdo.merge({
      status: 'signed',
      pdf: fileUrl,
    })

    await rdo.save()
  } catch (error) {
    console.error(error)
    throw error
  }
}
```

4.4 Manter, evoluir e disponibilizar um produto

4.4.1 Projeto D

Como citado em etapas anteriores, a ioasys possui um produto interno, que é o Projeto D, o qual foi onde o estagiário foi alocado após o término de suas atividades no projeto C.

Esta foi a etapa mais marcante de todo o processo de estágio para o estagiário, pois, foi onde ele ganhou mais responsabilidades em comparação às etapas anteriores.

O projeto D, tratando-se de um produto, é utilizado por vários clientes da ioasys. O estagiário foi responsável por manter as APIs de todos os clientes atualizadas e funcionando, cada uma hospedada no respectivo servidor dedicado do cliente.

Assim como em outros projetos, no Projeto D o estagiário desenvolveu novos *endpoints*, assim como efetuou manutenções em *endpoints* existentes. Sempre que um *endpoint* era criado ou atualizado, o código passava por um processo de avaliação e aprovação dos outros desenvolvedores da equipe de backend da ioasys e, mediante aprovação, era submetido a ramificação (branch) principal de desenvolvimento do repositório do Projeto D. Com a branch principal de desenvolvimento atualizada, a próxima responsabilidade do estagiário era atualizar a API no servidor de homologação com o novo código, disponibilizando a versão mais atual, contendo as novas funcionalidades e alterações para que os desenvolvedores *frontend* do Projeto D pudesse utilizá-la.

Após um determinado período, quando a versão da API que estava em homologação já estava testada e validada, a responsabilidade do desenvolvedor era atualizar a branch principal do repositório (master) com o código validado, e, a partir daí, gerar uma *release* (versão) para cada cliente. Feito isso, a próxima etapa era atualizar as APIs de todos os clientes em seus respectivos servidores, fazendo com que as versões mais atuais das APIs ficassem disponíveis para as novas versões do Admin (*site*) e dos aplicativos (iOS e Android), disponibilizando a mais nova versão do sistema para todos os usuários.

Ainda atuando no Projeto D, o estagiário realizou a configuração de ambiente de dois novos clientes, essa configuração consistiu em: Acessar uma máquina EC2 da Amazon Web Services AWS), via protocolo Secure Shell (ssh); fazer toda a instação do ambiente Ruby on Rails, explicado em sessões anteriores; criação e configuração do banco de dados, utilizando Relational Database Service (RDS) da AWS; geração de certificado SSL para o domínio, que foi criado e disponibilizado por outro desenvolvedor,

via painel da AWS; configuração do Nginx, que era responsável por direcionar as requisições recebidas pelo domínio para a API disponibilizada no servidor, em uma determinada porta; configuração do Redis para armazenamento de dados em cache; entre outras.

As próximas figuras mostram alguns arquivos que compõe o projeto D:

- **CourseCertificate**: A (Figura 4.18) mostra o arquivo de *model* responsável pelo certificado de um curso. Sempre que um registro de certificado vai ser criado no banco, primeiro ele precisa passar pelas validações descritas nos métodos deste arquivo. Assim, para ser criado, o usuário precisa estar apto ao certificado (método user-is-entitled-to-certificate?) e ainda não ter recebido certificado para o curso em questão (método already-certificated?). Após passar por essas validações, o método send-certificate envia o certificado para o e-mail do usuário.

- **CourseSerializer**: A (Figura 4.19) mostra o arquivo de *serializer* responsável por montar o arquivo json de resposta contendo os dados de um curso. O campo attributes define quais campos da tabela serão exibidos no arquivo json. Os métodos definidos abaixo são campos a serem exibidos no arquivo json mas que não fazem parte da tabela cursos no banco de dados.

- **HardWorker**: A (Figura 4.20) mostra o arquivo que tem como objetivo enviar notificações para dispositivos *android* e *ios*.

- **NotificationSender**: A (Figura 4.22) mostra o arquivo de *service* que tem como objetivo enviar notificações para dispositivos *android*

e *ios*. O método get-tokens utiliza um outro arquivo de *service* chamado TokenSelector, que seleciona todos os *tokens* dos dispositivos de usuários, necessários no envio de notificações. O método create-push cria um registro na tabela de notificações no banco de dados. O método create-body cria o corpo da notificação, que será enviado para os dispositivos móveis da forma como eles o esperam receber. O método send-android-push envia notificações para dispotivos *android* e o método send-ios-push envia notificações para dispositivos *ios*. O método send-push faz a chamada dos dois últimos métodos.

- **sidekiq.yml**: A (Figura 4.23) mostra o arquivo de configuração do sidekiq [1], um serviço externo utilizado no projeto para o envio de *emails* e notificações. O arquivo define o agendamento de três tarefas. A primeira (canvas-publish-worker) é resposável por publicar uma postagem agendada para um determinado dia. Esta tarefa é executada todos os dias às 8:00. A segunda (log-cleaner-worker) é responsável por limpar todos os *logs* da API no servidor. Isso foi necessário para que eles não ocupassem muita memória física na máquina. Esta tarefa é executada todos os dias às 00:00. A terceira (raffle-worker) é responsável por realizar um sorteio agendado para um determinado dia. Esta tarefa é executada todos os dias às 8:00.

- **migrations**: A (Figura 4.24) mostra a execução dos comandos rails db:migrate e rails db:rollback. O primeiro comando realiza alterações no *schma* do banco de dados, que na figura foi exemplificado pela adição da coluna is-valid na tabela setores. O segundo comando

[1] https://sidekiq.org/

retrocede as últimas alterações feitas no *schema* pelo primeiro comando.

4.5 Mantendo e atualizando a API em um servidor

As próximas figuras mostram etapas comuns durante o processo de atualizar e manter a API nos servidores de diversos clientes.

- **Verificando se a API está executando**: A (Figura 4.25) mostra que a API está executando na porta 8005 do servidor.
- **Configuração de dominio no Nginx**: As (Figuras 4.26 e 4.27) mostram o conteúdo de um arquivo de configuração do Ngix [2] responsável por fazer as definições de como o Nginx vai receber as requisições e redirecionar para a aplicação.
- **Atualização de repositório**: A (Figura 4.28) mostra a atualização da branch master da API no servidor. Isso foi feito sempre que uma nova versão da API era lançada através de uma release.
- **Verificando se o sidekiq está executando**: A (Figura 4.29) mostra que o sidkiq está executando e utilizando o processo número 647 do ubuntu.
- **Status Nginx**: A (Figura 4.30) mostra que o Nginx está ativo e rodando sem problemas no servidor.

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

4.6 Ligação entre Teoria e Prática

Preenchendo uma vaga em uma empresa, o estagiário deve aproveitar para usar o que foi aprendido nas disciplinas durante o curso. Assim, faz-se uma ligação entre a teoria (disciplinas) e a prática (estágio).

Sabe-se que, no mercado de trabalho, existem diversas áreas em que o profissional pode atuar, mas assim como tudo na vida, é importante se especializar e focar na área de maior interesse e aptidão. Assim, parte do conhecimento aprendido nas disciplinas poderá ter conexão com a prática do periodo do estágio, em contrapartida, a outra parte pode não ter.

Para o estagiário, quatro disciplinas foram imprescindíveis para que ele estivesse pronto para atuar como estagiário backend na ioasys. Introdução aos algorítimos e Estruturas de dados deram a base tanto de lógica, como de programação e, isso contribuiu para que o estagiário não tivesse grandes dificuldades em pensar sobre resoluções de problemas. Desta forma, as maiores dificuldades encontradas foram nas tecnologias utilizadas que ele não vivenciou durante a graduação. Introdução ao sistema de bancos de dados e Sistemas gerenciadores de banco de dados forneceram vasto conhecimento teórico sobre modelagem de bancos de dados, consultas, *scrips* e transações. O estagiário acredita que isto tenha sido de grande ajuda e importância pois, como dito em sessões anteriores, o desenvolvimento de APIs está relacionado e é completamente dependente do uso de bancos de dados.

Assim, o estágio constitui possibilidades para que o graduando coloque em prática o conhecimento que foi absorvido cursando as disciplinas, tendo em vista que é importante assimilar os conhecimentos, científicos e empíricos, com a prática profissional.

Figura 4.11 – Projeto C - Criação de RDO

```
static async createRdo(rdoObject) {
  const {
    responsibleName,
    dateRdo,
    weatherCondition,
    rainfallIndex,
    workShift,
    enterpriseId,
    rdoDirectManpowers,
    rdoIndirectManpowers,
    rdoEquipments,
    rdoActivities,
  } = rdoObject
    Rodrigo Santos, a year ago • Generate migrations and models
  const trx = await Database.beginTransaction()

  const rdoNumber = await this.generateNumberRdo(dateRdo, enterpriseId)

  await this.validateNumberRdo(rdoNumber, trx)

  await this.validateDateRdo(enterpriseId, dateRdo)

  await this.checkRdoYesterday(enterpriseId, dateRdo)

  const rdo = new Rdo()

  rdo.fill({
    rdoNumber,
    responsibleName,
    dateRdo,
    weatherCondition,
    rainfallIndex,
    workShift,
    enterpriseId,
  })
```

```
  try {
    await rdo.save(trx)

    await RdoResourceService.createOrUpdateRdoDirectManpowers(
      rdoDirectManpowers,
      rdo.id,
      enterpriseId,
      trx
    )
    await RdoResourceService.createOrUpdateRdoIndirectManpowers(
      rdoIndirectManpowers,
      rdo.id,
      enterpriseId,
      trx
    )

    await RdoResourceService.createOrUpdateRdoEquipments(
      rdoEquipments,
      rdo.id,
      enterpriseId,
      trx
    )

    await RdoResourceService.createOrUpdateRdoActivities(
      rdoActivities,
      rdo.id,
      enterpriseId,
      trx
    )

    trx.commit()
    await AbnormalityService.associateAbnormalityRdo(
      moment(dateRdo, 'YYYY/MM/DD'),
      enterpriseId,
      rdo.id
    )
  } catch (error) {
    console.error(error)
    trx.rollback()
    throw error
  }
```

Figura 4.12 – Projeto C - Verificação de quantidade de dias sem assinar RDOs

```
static async checkEnterprisesRdos() {
  const today = moment()
  try {
    const enterprises = await Enterprise.all()
    for (let enterprise of enterprises.rows) {
      let foreseenStartDate = moment(
        enterprise.foreseenStartDate,
        'YYYY/MM/DD'
      )
      if (foreseenStartDate.isBefore(today, 'days')) {
        let missingRdosAmount = await RdoService.amountOfDaysWithoutRdo(
          enterprise
        )
        if (missingRdosAmount > 0) {
          await RdoService.notificateMissingRdos(
            missingRdosAmount,
            enterprise
          )
        }
      }
    }
  } catch (error) {
    console.error(error)
    throw 'Erro ao verificar número de Rdos não registrados dos empreendimentos.'
  }
}
```

Figura 4.13 – Projeto C - Sincronização de dados (declaração das tabelas de previsões)

```
const Database = use('Database')
const moment = use('moment')

const tablesOfResourceWithPrevisions = [
  'directManpowers',
  'indirectManpowers',
  'equipment',
]

const tablesOfPrevisions = {
  directManpowers: {
    tableName: 'directManpowerPrevisions',
    ref: 'directManpowerId',
  },
  indirectManpowers: {
    tableName: 'indirectManpowerPrevisions',
    ref: 'indirectManpowerId',
  },
  equipment: {
    tableName: 'equipmentPrevisions',
    ref: 'equipmentId',
  },
}
```

Figura 4.14 – Projeto C - Sincronização de dados (incluindo tabela de empreendimentos no vetor de diferenças)

```
class SynchronizationService {
  static async newDataSync(user) {
    await user.load('enterprises')
    const enterprises = user.toJSON().enterprises

    let diffs = []

    diffs.push({
      table: 'enterprises',
      new: enterprises,
      changes: [],
      deleted: [],
      unlinked: [],
    })

    const enterprisesIds = enterprises.reduce((ids, e) => {
      ids.push(e.id)
      return ids
    }, [])
```

Figura 4.15 – Projeto C - Sincronização de dados (incluindo dados de sincronização para a tabela de recursos com previsões)

```
for (let table of tablesOfResourceWithPrevisions) {
  const resources = await Database.table(table)
    .whereIn('enterpriseId', enterprisesIds)
    .where('deletedAt', null)

  diffs.push({
    table: table,
    new: resources,
    changes: [],
    deleted: [],
  })

  const resourcesIds = resources.reduce((ids, r) => {
    ids.push(r.id)
    return ids
  }, [])

  const previsions = await Database.table(
    tablesOfPrevisions[table].tableName
  )
    .whereIn(tablesOfPrevisions[table].ref, resourcesIds)
    .where('deletedAt', null)

  diffs.push({
    table: tablesOfPrevisions[table].tableName,
    new: previsions,
    changes: [],
    deleted: [],
  })
}
```

```
const activities = await Database.table('activities').whereIn(
  'enterpriseId',
  enterprisesIds
)

diffs.push({
  table: 'activities',
  new: activities,
  changes: [],
  deleted: [],
})

const contractualRisks = await Database.table('contractualRisks')
  .whereIn('enterpriseId', enterprisesIds)
  .where('deletedAt', null)

diffs.push({
  table: 'contractualRisks',
  new: contractualRisks,
  changes: [],
  deleted: [],
})

const abnormalities = await Database.table('abnormalities')
  .whereIn('enterpriseId', enterprisesIds)
  .where('deletedAt', null)

diffs.push({
  table: 'abnormalities',
  new: abnormalities,
  changes: [],
  deleted: [],
})

const abnormalitiesIds = abnormalities.reduce((ids, a) => {
  ids.push(a.id)
  return ids
}, [])
```

Figura 4.16 – Projeto C - Listagem de RDOs

```
async index({ request, response, auth }) {
  const query = request.all()
  const column = query.column || 'rdos.updatedAt'
  const direction = query.direction || 'desc'
  const page = parseInt(query.page) || 1
  const perPage = parseInt(query.perPage) || 10

  try {
    const user = await auth.getUser()
    await user.load('enterprises', builder => builder.setVisible(['id']))
    const enterprisesIds = user
      .toJSON()
      .enterprises.map(enterprise => enterprise.id)

    await Service.validateEnterprises(enterprisesIds)

    let rdos

    if (user.profileId !== 1) {      You, 10 months ago • improving synta
      rdos = await Rdo.query()
        .select(
          'rdos.id',
          'enterpriseId',
          'rdoNumber',
          'dateRdo',
          'enterprises.name AS enterpriseName',
          'status'
        )
        .innerJoin('enterprises', 'rdos.enterpriseId', 'enterprises.id')
        .whereIn('enterpriseId', enterprisesIds)
        .filter(query)
        .orderBy(column, direction)
        .paginate(page, perPage)
    } else {
      rdos = await Rdo.query()
        .select(
          'rdos.id',
          'enterpriseId',
          'rdoNumber',
          'dateRdo',
          'enterprises.name AS enterpriseName',
          'status'
        )
        .innerJoin('enterprises', 'rdos.enterprise_id', 'enterprises.id')
        .filter(query)
        .orderBy(column, direction)
        .paginate(page, perPage)
    }

    return response.status(200).json(rdos)
  } catch (err) {
    console.error(err)
    return response.status(400).json(err)
  }
}
```

Figura 4.17 – Projeto C - Notificação ao admin sobe os dias sem assinar RDOs

```
static async notificateMissingRdos(missingRdosAmount, enterprise) {
  try {
    const data = {
      type: 'missingRdos',
      objectId: enterprise.id,
      message: `A obra ${enterprise.name} possui ${missingRdosAmount} dias sem lançar RDOs`,
    }
    await NotificationService.sendNotificationForWeb({
      type: data.type,
      topic: 'notifications',
      data,
    })
  } catch (error) {
    console.error(error)
    throw 'Erro ao notificar Admin-ADCT e Admin-Escritório sobre a quantidade de dias sem lançar RDOs.'
  }
}       You, a year ago • Changing methods from service to another
```

Figura 4.18 – Projeto D - Certificado de curso

```
class CourseCertificate < ApplicationRecord
  belongs_to :course
  belongs_to :user
  validate :user_is_entitled_to_certificate?
  validate :already_certificated?
  after_create :send_sertificate

  def user_is_entitled_to_certificate?
    if !CertificateVerificator.new(self.user, self.course).perform or
        !self.course.user_already_finished?(self.user)
      errors.add(:base, 'Usuário não está apto ao certificado.')
    end
  end

  def send_sertificate
    CourseMailer.send_email_with_certificate(self.user, self.course).deliver_now      You, 21
  end

  def already_certificated?
    if self.user.present? && self.course.present?
      if CourseCertificate.find_by(user_id: self.user.id, course_id: self.course.id).present?
        errors.add(:base, 'Usuário já foi certificado')
      end
    end
  end
end
```

Figura 4.19 – Projeto D - Serializer de curso

```ruby
class CourseSerializer < ActiveModel::Serializer
  belongs_to :module_courses
  attributes :id, :title, :description,
             :image, :score, :duration,
             :level, :module_courses,
             :tags,:available, :minimum_percentage_for_certificate,
             :finished_modules, :users_progress_data, :size

  def tags
    object.tags
  end

  def finished_modules
    object.module_courses.pluck(:id) &
    current_user.validate_modules.pluck(:module_course_id).map(&:to_i)
  end

  def size
    @instance_options[:size]
  end

  def users_progress_data
    return unless current_user.has_role? :admin

    users = User.select(:id,:name,:image).joins(:tags).where(tags: { name: object.tags_scope })

    did_not_finish = []
    finished = []

    users.each do |user|
      if object.user_already_finished?(user) || object.validate_courses.find_by(user_id: user.id).present?
        finished << user
      else
        user_validate_modules_ids = user.validate_modules.pluck(:module_course_id).map(&:to_i)
        course_finished_modules = object.module_courses.pluck(:id) & user_validate_modules_ids
        did_not_finish << {
          user: user,
          finished_modules: course_finished_modules
        }
      end
    end

    {
      users_who_finished: finished,
      users_who_did_not_finish: did_not_finish
    }
  end
end
```

Figura 4.20 – Projeto D - Envio de notificações

```ruby
class HardWorker
  include Sidekiq::Worker
  # ('Temos uma nova publicação', authentication.metadata[:device_token],device='android',create_body_notification(user))
  def perform(*args)
    if args[1].is_a? Array
      if args[2] == 'ios'
        send_all_ios_push(args[0], args[1], args[2], args[3], args[4], args[5])
      else
        send_all_android_push(args[0], args[1], args[3])
      end
    else
      if args[2] == 'ios'
        send_ios_push(args[0], args[1], args[2], args[3], args[4], args[5])
      else
        send_android_push(args[0], args[1], args[3])      You, 3 months ago • refactoring push_canvas
      end
    end
  end

  def send_ios_push(message, device_token, _device, bodynotification, _user, kindpush)
    notification = Houston::Notification.new(device: device_token)
    notification.mutable_content = 1
    notification.category = kindpush
    notification.alert = message
    notification.custom_data = bodynotification unless bodynotification.nil?
    APN.push(notification)
  end

  def send_android_push(_message, device_token, body)
    FCM_SENDER.send([device_token], data: body)
  end

  def send_all_android_push(_message, device_token, notification_data)
    FCM_SENDER.send(device_token, { priority: 'high', data: notification_data })
  end
```

```ruby
  def send_all_ios_push(message, device_tokens, _device, notification_data, _user, kindpush)
    device_tokens.each do |token|
      notification = Houston::Notification.new(data: notification_data,
                                               device: token,
                                               mutable_content: 1,
                                               category: kindpush,
                                               alert: message
                                               )

      APN.push(notification)
    end
  end
end
```

Figura 4.21 – Projeto D - Verificação de aptdão para certificado

```ruby
class CertificateVerificator
  def initialize(current_user, course)
    @user = current_user
    @course = course
  end

  def perform
    check_certificate_availability
  end

  private

  def check_certificate_availability
    if @course.module_courses
                .joins(contents: :validate_answer_repeateds)
                .where('validate_answer_repeateds.is_correct = ?', true)
                .where('validate_answer_repeateds.user_id = ?', @user.id).size >=
        @course.minimum_amount_of_correct_responses_to_get_certificate
      true
    else
      false
    end
  end
end      You, 2 months ago • created new service to verificate certificate avai
```

Figura 4.22 – Projeto D - Serviço de envio de notificações

```ruby
class NotificationSender
  def initialize(users, notification_params, current_user)

    @current_user = current_user
    @notification_params = notification_params
    @users_to_send = users
  end

  def perform
    get_tokens
    create_push
    create_body
    send_push
  end

  private
  def get_tokens
    @android_tokens, @ios_tokens = TokenSelector.new(@users_to_send).perform
  end

  def create_push
    @users_to_send.each do |user|
      Notification.create(title: @title, description: @description, kind: 'alert')
    end
  end

  def create_body
    @body = {
      title: @notification_params[:title],
      message: @notification_params[:description],
      type: 'alert',
      sending_user: {
        name: @current_user.name,
        image: (
          @current_user.image.thumb.url.nil? ?
          '' :
          @current_user.image.thumb.url), id: @current_user.id}
    }
  end
```

```ruby
  def send_push
    send_android_push
    send_ios_push
  end

  def send_android_push
    HardWorker.perform_async(
      'Alerta de Notificação',
      @android_tokens,
      'android',
      @body,
      @current_user
    )
  end

  def send_ios_push
    HardWorker.perform_async(
      'Alerta de Notificação',
      @ios_tokens,
      'ios',
      @body,
      @current_user,
      'alert'
    )
  end
end
```

Figura 4.23 – Projeto D - Definição e agendamento de tarefas

```
:schedule:      You, 2 months ago • crea
  canvas_publish_worker:
    cron: "0 8 * * * America/Sao_Paulo"
    class: CanvasPublishWorker
  log_cleaner_worker:
    cron: "0 0 * * * America/Sao_Paulo"
    class: LogCleanerWorker
  raffle_worker:
    cron: "0 8 * * * America/Sao_Paulo"
    class: RaffleWorker
```

Figura 4.24 – Projeto D - Execução e rollback de migration

```
→ [redacted]backend-cliente (feature/user-course-data)rails db:rollback
== 20200629110057 AddValidToSectors: reverting ================================
-- remove_column(:sectors, :is_valid, :boolean, {:default=>true})
   -> 0.1735s
== 20200629110057 AddValidToSectors: reverted (0.1786s) =======================

→ [redacted]backend-cliente (feature/user-course-data)
→ [redacted]backend-cliente (feature/user-course-data)rails db:migrate
== 20200629110057 AddValidToSectors: migrating ================================
-- add_column(:sectors, :is_valid, :boolean, {:default=>true})
   -> 0.1420s
== 20200629110057 AddValidToSectors: migrated (0.1422s) =======================
```

Figura 4.25 – Projeto D - API executando na máquina

```
ubuntu@[redacted]:~$ ps -ef | grep puma
ubuntu      624     1  0 Jun26 ?        00:05:51 puma 3.10.0 (tcp://0.0.0.0:8005) [[redacted]backend-cliente]
ubuntu    14709 14569  0 11:42 pts/1    00:00:00 grep --color=auto puma
ubuntu@[redacted]~$
```

Figura 4.26 – Projeto D - Arquivo de configuração de aplicação no Nginx

Figura 4.27 – Projeto D - Arquivo de configuração de aplicação no Nginx

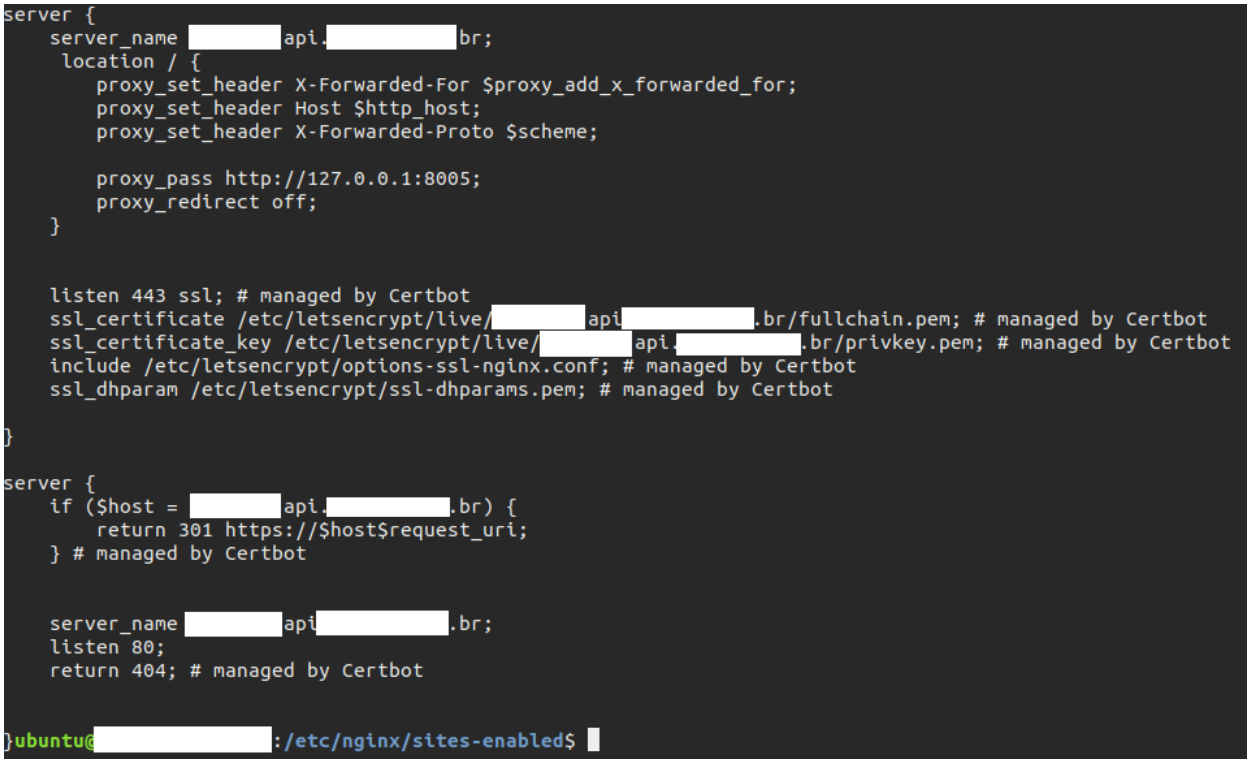


Figura 4.28 – Projeto D - Atualização do código da API

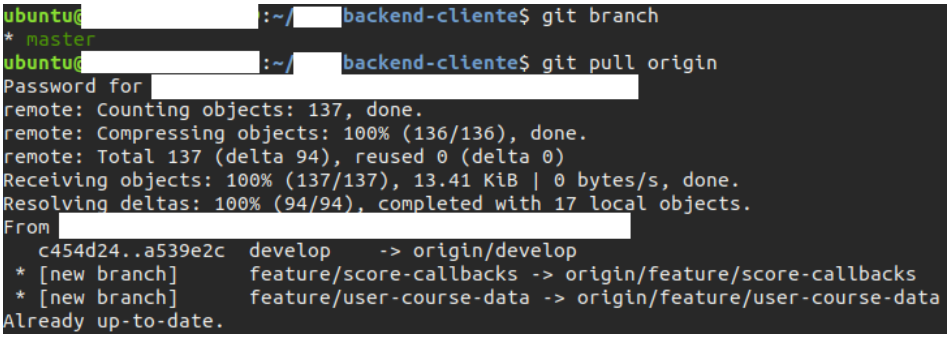


Figura 4.29 – Projeto D - Sidekiq rodando na máquina

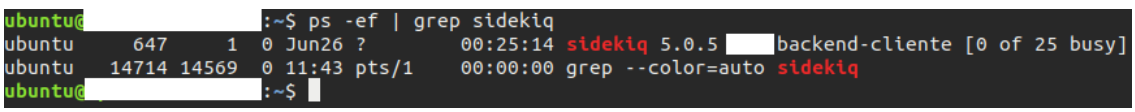
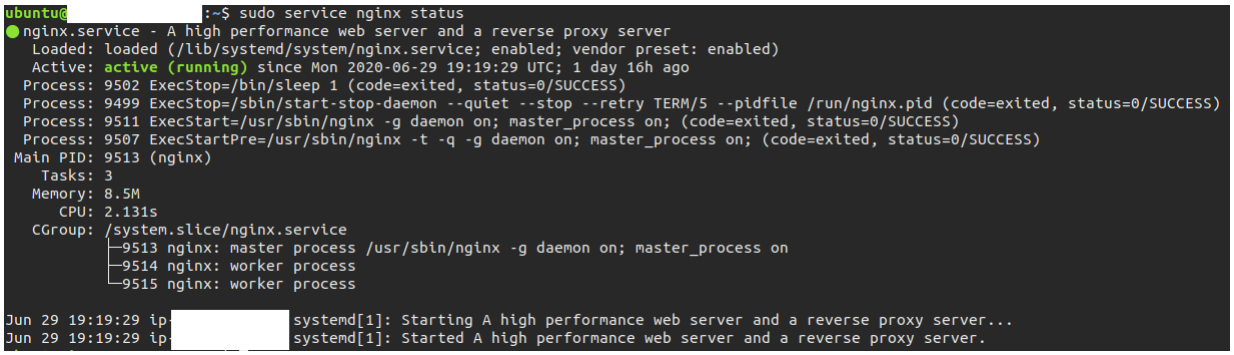


Figura 4.30 – Projeto D - Status do Nginx



REFERÊNCIAS

CARVALHO, V. **PostgreSQL: Banco de Dados para Aplicações Web Modernas**. [S.l.]: Casa Do Código, 2017.

ESPINHA, R. G. **Kanban: o que é e TUDO sobre como gerenciar fluxos de trabalho**. 2019. (Acessado em 28/05/2019). Disponível em: <https://artia.com/kanban/>.

SCRUM. 2019. (Acessado em 29/05/2019). Disponível em: <https://www.desenvolvimentoagil.com.br/scrum/>.