JOAB WENDSON RIBEIRO SANTOS

DESENVOLVIMENTO DE MICROSSERVIÇOS NA RAISE SISTEMAS

LAVRAS – MG

2022

JOAB WENDSON RIBEIRO SANTOS

DESENVOLVIMENTO DE MICROSSERVIÇOS NA RAISE SISTEMAS

Relatório de estágio supervisionado 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. Dilson Lucas Pereira
Orientador

LAVRAS – MG

2022

JOAB WENDSON RIBEIRO SANTOS

DESENVOLVIMENTO DE MICROSSERVIÇOS NA RAISE SISTEMAS

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

APROVADA em 06 de Setembro de 2022.

Prof. Dr. RAFAEL SERAPILHA DURELLI UFLA
Prof. Dr. ANDRE VITAL SAUDE UFLA

Prof. Dr. Dilson Lucas Pereira
Orientador

LAVRAS – MG
2022

Dedico este trabalho à toda minha família. Em especial aos meus pais e a minha noiva que estiveram comigo durante o trajeto da graduação.

AGRADECIMENTOS

Agradeço primeiramente à Deus por sempre está comigo e me dar força e saúde para conseguir concluir este curso.

Aos meus pais por sempre acreditar em mim nos momentos mais difíceis e me mostra o caminho dos estudos, sempre me incentivando e encorajando.

A minha noiva por sempre me apoiar nos momentos mais difíceis e por está sempre ao meu lado.

Aos meus amigos por todos os momentos divertidos e contagiantes.

Ao meu orientador Dilson, por todo apoio na realização deste trabalho e aprendizados durante o projeto que trabalhamos juntos.

A Universidade Federal de Lavras e todo o seu corpo docente, que foi essencial no meu processo de formação profissional.

RESUMO

Nos últimos tempos, a arquitetura de microsserviços está sendo cada vez mais utilizada por empresas de referência. Sua flexibilidade permite a utilização de diversas linguagens de programação no mesmo projeto o que o torna mais escalável gerando um aumento na produtividade e segurança. Este documento relata as atividades desenvolvidas durante um estágio na empresa Raise Sistemas LTDA, que possui duas startups: Atos6 e E-Inscrição. A primeira é um sistema de gerenciamento financeiro de igrejas, já a segunda é uma plataforma de gestão de eventos e cursos. Para o sistema financeiro de tais empresas foram construídos microsserviços de pagamentos via pix, cartões de crédito e boletos bancários. Esses pequenos serviços foram produzidos utilizando a linguagem de programação TypeScript e as APIs(Interface de programação de aplicações) de sistemas de pagamento. Os microsserviços foram hospedados na plataforma Cloudflare Worker.

Palavras-chave: Microsserviços, Linguagens de programação, TypeScript, Interfaces de Programação de Aplicações, Cloudflare Worker.

ABSTRACT

In recent times, the use of microservices architecture is being increasingly used by leading companies. Its flexibility allows the use of several programming languages in the same project, which makes it more scalable, generating an increase in productivity and security. This document reports the activities developed during an internship at the company Raise Sistemas LTDA, which has two startups: Atos6 and E -Inscrição. The first is a church financial management system, the second is an event and course management platform. For the financial system of such companies, microservices were built for payments via pix, credit card and bank slips. These small services were produced using the TypeScript programming language and the APIs (Application Programming Interface) of payment systems. The microservices were hosted on the Cloudflare Worker platform.

Keywords: Microservices, Programming Languages, TypeScript, Application Programming Interfaces, Cloudflare Worker.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A Raise Sistemas surgiu como uma empresa para solucionar problemas de gestão de igrejas. Começou com um *software* pequeno e em seguida, com o seu crescimento, foi necessário a criação de uma *startup*, a Atos6. Com o desenvolvimento da Atos6 surgiu um novo problema, a organização dos eventos das igrejas, que até então era realizada de forma manual. Logo, como solução, surgiu a plataforma do E-inscrição que tinha como objetivo a resolução desse problema. Com o passar do tempo, ela abraçou todos os tipos de eventos, se tornando uma *startup* também. A maioria das funções que as duas plataformas apresentam está construída em uma arquitetura monolítica, porém algumas funções são construídas com uma arquitetura de microsserviços.

Um sistema monolítico é um modelo composto por uma única peça, ou seja, todas as funções de negócio são implementadas no mesmo código fonte e unidade de instalação. Isso acarreta no compartilhamento de recursos comuns entre as funções e na dependência entre elas (WOODS, 2015).

Na plataforma da Atos6 umas das soluções oferecidas é a possibilidade do usuário realizar doações online, e em cada transação via cartão de crédito ou boleto bancário concluída era necessário pagar uma taxa para a empresa que fornecia o *gateway* de pagamento. Empresas que fornecem esses serviços geralmente cobram por cada transação concluída, cada uma delas possui taxas diferentes, sendo mais barata ou mais cara, dependendo do serviço escolhido. Porém, o valor cobrado se tornou inviável para a Atos6. Diante disto, foi necessária a troca do serviço de pagamento que até então estava sendo utilizado, sendo necessária a construção do primeiro microsserviço, responsável por realizar transações via cartão de crédito e geração de boletos bancários, utilizando outro *gateway* de pagamento.

Na plataforma do E-Inscrição o pagamento dos usuários para participar dos eventos podiam ser realizados pela própria plataforma, porém não era possível realizar pagamentos via pix, e desde o lançamento deste sistema de pagamento, criado pelo Banco Central, houve uma aderência muito grande por parte dos usuários (BRASIL, 2022). Nesse cenário, viu-se a necessidade de implementar essa nova forma de pagamento na plataforma do E-inscrição, sendo necessário a construção de um segundo microsserviço, que tinha a responsabilidade de realizar transações via pix.

O objetivo deste trabalho foi desenvolver dois microsserviços para atender a migração de um *gateway* de pagamento e a necessidade da implementação do pix.

O estágio foi realizado entre fevereiro e agosto de 2021, motivado pela oportunidade de aprendizado e crescimento pessoal. O estagiário contou com uma equipe composta por três desenvolvedores *back-ends* e três desenvolvedores *front-ends* junto com o *tech Lead*, com uma carga horária semanal de 30 horas. O estagiário participou do desenvolvimento de dois microsserviços durante este período, atuando como desenvolvedor *back-end*.

Back-end está relacionado com o que vem por trás de uma aplicação, ele é armazenado e executado em um servidor é responsável por dar estrutura e apoio às ações do usuário. Logo, ele geralmente trabalha fazendo a ponte entre o que vem do navegador com o que está no banco de dados e vice-versa, essa ponte é onde se aplica as regras de negócio do sistema. Portanto, o desenvolvedor *back-end* é responsável por implementar e garantir que essa comunicação ocorra da forma esperada e com segurança.

1.1 Empresa

A Raise Sistemas é uma empresa do Rio de Janeiro que tem como função gerenciar o produto das duas startups: a Atos6 e o E-Inscrição. Sendo que o produto da Atos6 é uma plataforma de gestão de igrejas online, oferecendo diversas soluções de gestão como célula, financeiro, doação online, eventos e culto online. Ela é a escolhida por mais de 4 mil igrejas como a solução para os seus problemas de gestão e possui cerca de 250 mil usuários.

Já o E-Inscrição é uma startup especializada em gestão de eventos, possuindo uma plataforma online que oferece diversos recursos como emissão de certificados, checking de usuários, pagamento pela própria plataforma e facilita a comunicação entre o organizador e os inscritos com envio de emails. Ela atua desde a inscrição até o certificado, passando pela comunicação, financeiro e pelo check-in. A plataforma possui mais de 16 mil organizadores de eventos cadastrados e já possui mais de 1,5 milhões de inscrições realizadas.

A Raise Sistemas possui cerca de 60 colaboradores, que são divididos entre os dois produtos gerenciados por ela. Todos os colaboradores trabalham de forma remota, esse modelo de trabalho é idealizado pela própria instituição, visando o bem estar e comodidade de todos . A empresa é dividida em diversos setores, entre estes o estagiário foi alocado no setor *tech*, responsável por todo desenvolvimento dos produtos. Neste setor, tinham 12 desenvolvedores e o *tech lead*, sendo 6 desenvolvedores alocados para o desenvolvimento dos microsserviços, sendo dividido igualmente entre desenvolvedor front-end e back-end. Os outros 6 deles, com mais experiência, são responsáveis pelo sistema monolítico da Atos6 e E-Inscrição. O *tech lead* é responsável por todo o setor.

1.2 Organização do Documento

O presente documento está dividido em:

- Capítulo 2, especifica os conceitos e tecnologias utilizadas durante todo o processo;
- Capítulo 3, aborda o processo de desenvolvimento durante o período de estágio;
- Capítulo 4, aborda as atividades executadas durante o período de estágio;
- Capítulo 5, temos a conclusão do desenvolvimento das atividades durante o período de estágio.

2 CONCEITOS E TECNOLOGIAS

Este capítulo apresenta as tecnologias e os conceitos utilizados na construção dos microsserviço de pagamento durante o período de estágio.

2.1 Git

No desenvolvimento de um *software* é necessário se ter um controle de versões para que toda a equipe possa trabalhar no mesmo projeto sem causar inconsistências. Além disso, é importante que se tenha a capacidade de armazenar diferentes versões do projeto para evitar perdas. O controle de versão é um sistema para armazenar diferentes etapas de um projeto ao longo do tempo, e assim, permitir o acesso de versões específicas ao usuário posteriormente(CHACON, 2014).

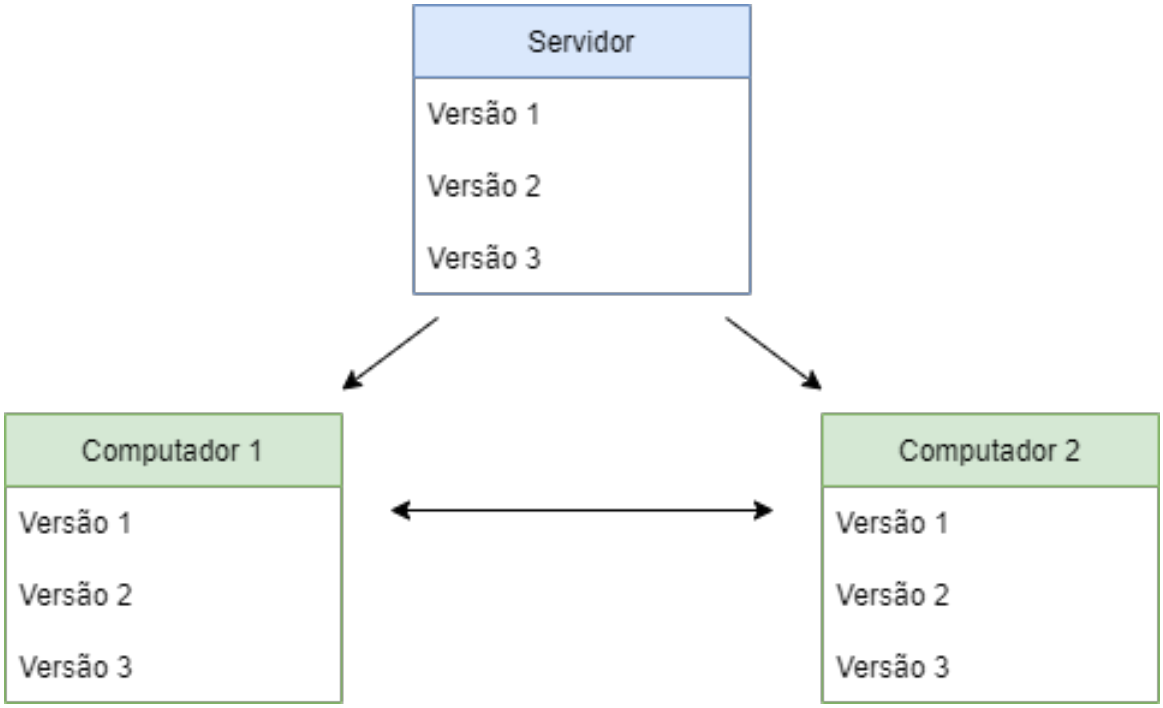
Esse processo é muito importante durante o desenvolvimento, pois permite que vários programadores trabalhem no mesmo projeto sem que haja sobrescrita de código um do outro, trazendo um maior grau de segurança para todo o processo de desenvolvimento.

Existem três categorias principais no controle de versão: Sistemas Locais de Controle de Versão (SCV), Sistemas Centralizados de Controle de Versão (CVCSs) e Sistemas Distribuídos de Controle de Versão (DVCS). O primeiro consiste em armazenar as diferentes versões de um projeto em um banco de dados simples localmente, o problema dele é que impossibilita que outros usuários em sistemas diferentes possam colaborar entre si. O segundo, permite a colaboração entre os usuários, pois possui um servidor central que contém o sistema de controle de versão com os arquivos versionados do projeto, porém o usuário possui apenas uma cópia do arquivo localmente, as versões ficam armazenadas apenas no servidor, isso se torna um problema caso o servidor fique *offline* ou seja corrompido. O terceiro, é um sistema em que os usuários possuem uma cópia exatamente igual a do servidor, resolvendo o problema do controle de versão centralizado (CHACON, 2014).

O GIT é um sistema de controle de versão distribuído, gratuito e de código aberto, com ele é possível de forma simples buscar versões antigas do projeto e manter a consistência do código desenvolvido em todas as etapas do processo. Além disso, um grande benefício do GIT é o fato de ser distribuído, com isso, em cada repositório, incluindo o da máquina do contribuidor, vai existir uma cópia completa e funcional, como ilustrado na Figura 2.1. Assim, se o servidor ficar corrompido a cópia do repositório vai estar intacta localmente, e o usuário pode restaurar completamente o histórico completo do que foi desenvolvido, evitando que os dados

sejam perdidos (ROVEDA, 2021). Todas as atividades realizadas no estágio foram construídas utilizando o GIT para o versionamento de código e os repositórios foram hospedados por meio da plataforma GitHub[1].

Figura 2.1 – Controle de versão distribuido



Fonte: elaborada pelo autor

2.2 Interface de Programação de Aplicação (API)

API é um mecanismo que possibilita a comunicação de mais de um sistema, compartilhando assim, informações entre si (SERVICES, 2022). Atualmente existem quatro formas mais utilizadas em que as APIs podem ser criadas, que são SOAP (Protocolo de Acesso a Objetos Simples), RPC (Chamadas de Procedimento Remoto), WebSocket e REST. Essa última, é a mais usada atualmente em serviços web.

2.3 Arquitetura de Microsserviços

Com o avanço da tecnologia, nos últimos anos, empresas como Amazon, Netflix, Twitter e Uber, têm adotado uma rápida mudança que vem se espalhando pelo mundo, a adoção de metodologias distribuídas, arquitetura de microsserviços e a construção de sistemas monolíticos. Isso se dá, principalmente, para sanar os problemas e desafios de escalabilidade, falta

[1] <https://github.com>

de eficiência, problemas no desenvolvimento e a dificuldade em adotar novas tecnologias no projeto. Geralmente a aplicação de uma empresa que está iniciando no mercado é pequena e de baixa complexidade, porém conforme a empresa cresce a aplicação vai acompanhando esse ritmo, e com isso há aumento na complexidade que ocorre devido ao aumento das funcionalidades. Com esse crescimento, a adição e ajuste nas funcionalidades também se torna difícil, pois devido ao seu grau de acoplamento pode fazer com que outras funcionalidades deixem de funcionar como deveriam, acarretando outros problemas (FOWLER, 2017).

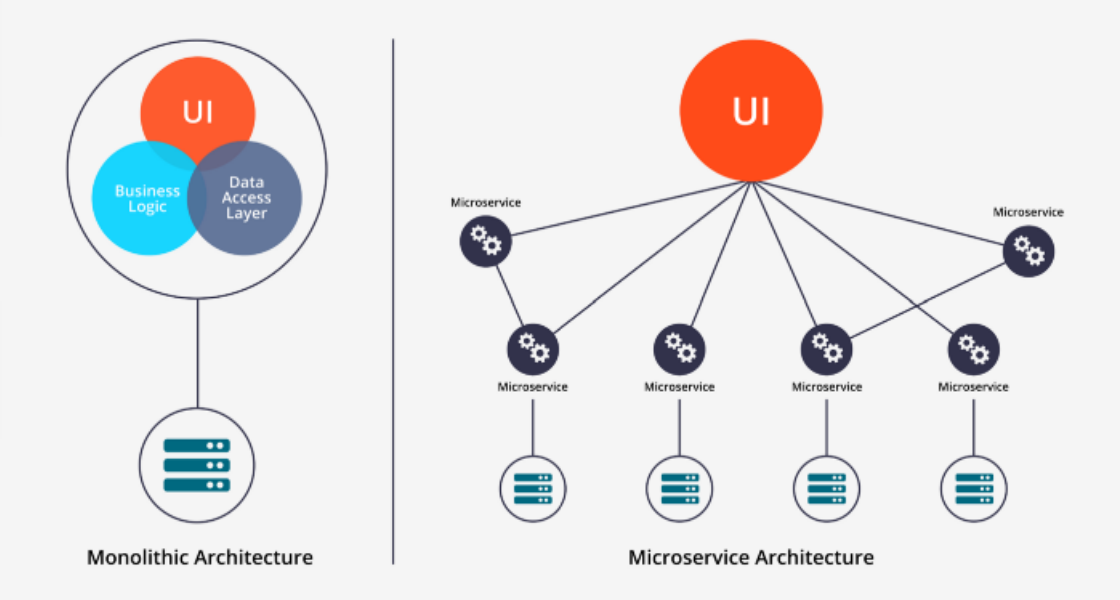
Microsserviços é uma abordagem arquitetônica e organizacional em que vários pequenos serviços desempenham funções específicas. Logo, como as tarefas são divididas em cada serviço elas são fracamente acopladas, isso garante uma independência no desenvolvimento dos serviços (DAYA et al.,).

Em uma arquitetura de microsserviços é comum que APIs sirvam como ponto de interação (endpoints) entre o usuário que acessa o sistema e o microsserviço. Eles são pontos de acesso estáticos dentro do sistema para percorrer um determinado fluxo. Nos microsserviços desenvolvidos neste trabalho, foi utilizado endpoints para referenciar cada fluxo a ser percorrido de acordo com a solicitação do usuário.

A diferença entre a arquitetura monolítica e a arquitetura de microsserviços está na forma em que as regras de negócios são implementadas, enquanto que na monolítica todas as regras de negócios são implementadas em um único processo, a de microsserviços separa cada uma de forma lógica em cada serviço independente, fazendo com que cada uma tenha um processo para ela. Algumas vantagens do uso dos microsserviços está na liberdade tecnológica, maior capacidade de escalabilidade e na resiliência do sistema.

Durante todo o processo de estágio, o desenvolvimento foi baseado na arquitetura de microsserviços, por conta da escalabilidade e facilidade tecnológica que ela proporcionava. A Figura 2.2 abaixo ilustra um exemplo da arquitetura monolítica e de microsserviços ideal. Pode-se observar algumas diferenças na implementação da arquitetura monolítica tradicional e arquitetura baseada em microsserviços. Na Raise Sistemas, os microsserviços relatados neste documento não tinham banco de dados, pois eles eram responsáveis apenas por realizar a comunicação com as plataformas de transações financeiras.

Figura 2.2 – Exemplo de arquitetura de microsserviços e monolítica



Fonte: (MALAV, 2017)

2.4 JavaScript

JavaScript é uma linguagem de programação leve, interpretada, dinâmica e multi-paradigma, conhecida como linguagem de *script*. É uma linguagem fracamente tipada, ou seja, não é necessário definir tipos das variáveis e objetos. Ela permite a criação de páginas web interativas, o que fez com que ela se tornasse essencial em aplicativos web mais robustos (FLANAGAN, 2013). Porém, com o passar dos anos ela começou a ser utilizada no desenvolvimento *back-end*, por meio do Node JS, que é um *software* de código aberto, multiplataforma que permite a execução do JavaScript fora de um navegador web. Por ser uma linguagem multi-paradigma, ela suporta estilos de orientação a objeto, imperativos e declarativos.

2.5 TypeScript

Desenvolvida pela Microsoft, o TypeScript é uma linguagem de código aberto construida a partir do JavaScript. A sua principal inovação é a tipagem e o suporte maior em relação ao uso da programação orientada a objeto em relação a linguagem original. Com uma tipagem maior é possível detectar erros no código sem executá-los, ou seja, é uma validação de código em tempo de execução (FENTON, 2018).

A Figura 2.3 ilustra uma função chamada soma, construída em JavaScript, ela tem a responsabilidade de retornar a soma de dois valores que são passados por parâmetro. Entretanto,

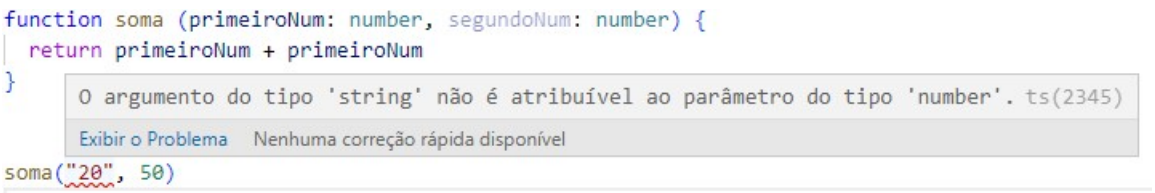
na chamada da função, o primeiro parâmetro é passado com o tipo *string*, e isso vai fazer com que se tenha uma concatenação em vez da soma dos valores, esse erro passa despercebido no JavaScript. Contudo, o mesmo código em TypeScript, como ilustrado na Figura 2.4, exibe um erro indicando a incompatibilidade de tipo. Logo, é possível perceber como essa inovação torna o código mais seguro e claro para o desenvolvedor.

Figura 2.3 – Código em JavaScript

```
function soma(primeiroNum, segundoNum){
 return primeiroNum + primeiroNum
}
soma("20", 50)
```

Fonte: elaborada pelo autor

Figura 2.4 – Código em TypeScript



Fonte: elaborada pelo autor

2.6 Cloudflare Worker

A Cloudflare é uma empresa que fornece serviços para melhorar desempenho e segurança a aplicações web. Seu recurso principal é o CDN (rede de distribuição de conteúdo), que consiste em uma rede de servidores distribuídos ao redor do mundo de forma estratégica permitindo um acesso rápido e segurança a serviços que usam essa ferramenta (CLOUDFLARE, 2022b). Ela também possui um serviço chamado Cloudflare Worker que utiliza desse serviço CDN e permite a implantação de serviços em nuvem de forma prática e simples, garantindo o desenvolvedor preocupação apenas com código e lógica em si, entregando a segurança necessária juntamente com os serviços essenciais por este de forma simples. (CLOUDFLARE, 2022a). Além disso, devido a utilização do CDN, quando uma aplicação é implantada ela é replicada nos diversos servidores da Cloudflare. Logo, quando é feita uma solicitação de acesso a algum serviço Worker, a Cloudflare escolhe o servidor mais próximo do usuário e faz o redirecionamento para ele. Isso faz com que a latência do serviço hospedado caia drasticamente, mesmo que seja acessada em outro país. Ele também possui um custo mais baixo em relação a outros

serviços, e isso foi um fator decisivo para o uso desse serviço nos microsserviços relatados neste documento.

A Figura 2.5 mostra um exemplo de estruturação de código para desenvolvimento no Cloudflare Worker de um serviço que retorna um *hello word*. Quando uma solicitação ao subdominio ou domínio gerenciado pela Cloudflare é recebida a ferramenta Worker recebe um argumento *FetchEvent* e envia para o manipulador de eventos, que como observado na imagem, é o *addEventListener*. Ao receber esse evento chama a função *eventListner*, que tem a responsabilidade de responder chamando a função *routerHandler*, responsável por fazer o controle das rotas de acordo com o *path* recebido.

Figura 2.5 – Exemplo de código simples que é aceito pelo Cloudflare Worker

```
function helloWord (event: FetchEvent){ // função para retornar um hello word
  if(event.request.method === 'GET') {
    try {
      return new Response('hello word', { status: 200 });
    } catch(error) {
      return new Response('Erro não seperado', { status: 500 });
    }
  } else {
    return new Response('Esperado GET', { status: 403 });
  }
}

const routes: any = { //objeto que faz o mapeamento dos paths.
  "/hello-word" : helloWord
}

// função para direcionar a função a ser executada de acordo com o path recebido.
function routerHandler(event: FetchEvent) {
  const url = new URL(event.request.url);
  const route = routes[url.pathname];
  return route(event);
}

function eventListner (event: FetchEvent) { //responsável por responder o evento recebido.
  event.respondWith(routerHandler(event));
}

addEventListener('fetch', eventListner); //manipulador de eventos.
```

Fonte: elaborada pelo autor

O *deploy* de uma aplicação é quando uma aplicação é colocada online, podendo ser utilizada por outros usuários. Para realizar *deploy* do microsserviço desenvolvido de forma automática na Cloudflare, é necessário ter um arquivo chamado *wrangler.toml* na raiz do projeto. Nele, é definido a configuração de desenvolvimento para a publicação de um Worker em ambiente de desenvolvimento. A Figura 2.6 abaixo ilustra um exemplo do conteúdo desse arquivo para o exemplo *hello word*. O nome do microsserviço é concatenado com a url base definida na plataforma do Cloudflare, gerando a url de acesso. Supondo que a url base definida seja *service.workers.dev*, ao realizar o *deploy* da aplicação o acesso externo do serviço seria por *hello-word.service.workers.dev*, e para ter acesso ao *endpoint* do fluxo implementado seria por *https://hello-word.service.workers.dev/hello-word*.

Figura 2.6 – Exemplo do arquivo de configuração

wrangler.toml

```
1   # nome do microsserviço
2   name = "hello-word"
3
4   #tipo de linguagem utilizada
5   type = "javascript"
6
7   #id da conta que receberá a aplicação
8   account_id = "af14559446db12d75e31175b11f2f9ff"
9
10  # indica ambiente de desenvolvimento
11  workers_dev = true
12
```

Fonte: elaborada pelo autor

2.7 JSON Web Token (JWT)

O token JWT é uma assinatura digital formada por uma sequência de caracteres. O JSON Web Token é um padrão baseado no formato JSON para autenticação e transmissão segura de dados, sendo que há duas formas de utilização para essa ferramenta. A primeira é para troca de informações, os JWT podem ser assinados com chave pública e privada, por exemplo. A segunda é para autorização, e foi a utilizada nos microsserviços implementados. Quando o usuário é conectado a um serviço, ele recebe um *token* que é enviado em cada solicitação para o recurso requisitado, podendo ter ou não um tempo de expiração (AUTH0, 2013). Na Figura 2.7 abaixo mostra um exemplo de como criar um *token* JWT na linguagem JavaScript com um tempo de expiração definido. Na Figura 2.8 ilustra um exemplo de verificação do *token*

gerado, nela é possível observar que é necessário passar uma chave secreta por parâmetro. Ela é necessária para sua decodificação e validação.

Figura 2.7 – Gerar Token JWT

```
const jwt = require('jsonwebtoken')
function gerarTokenJwt(){
  const token = jwt.sign({
    data: 'foobar'
  }, 'chaveSecreta', { expiresIn: 60 * 60 });

  return token;
}
```

Fonte: elaborada pelo autor

Figura 2.8 – Verificar Token JWT

```
const jwt = require('jsonwebtoken')
function verificarTokenJwt(token){
  const decodificado = jwt.verify(token, 'chaveSecreta');
  return decodificado;
}
```

Fonte: elaborada pelo autor

2.8 LogFlare

O LogFlare é uma ferramenta para registro de eventos, sendo possível registrar data, hora, origem da atividade, dados recebidos e enviados. Também tem a capacidade de armazenar e analisar esses dados a longo prazo (MAROTEL, 2021).

2.9 Gateway de Pagamento

Gateway de Pagamento é uma solução que serve como ponte para pagamentos digitais entre consumidor, cliente, bancos e operadoras de cartões de crédito. Isso possibilita a comunicação do negócio com os adquirentes, e com isso, mantém centralizada toda a comunicação de pagamento ao respectivo negócio. Essa solução é responsável por garantir a transmissão dos dados e armazenamento de forma segura e rápida (VINDI, 2017). Basicamente o *gateway* funciona da seguinte forma, quando um cliente realiza uma compra e envia os dados necessários para a plataforma, todas as informações são enviadas para um *gateway* de pagamento, que

realiza os procedimentos necessários para que o pagamento seja aprovado, garantindo a segurança de todas as informações. Com o pagamento aprovado, é retornando para a plataforma a confirmação compra, que posteriormente, é informada para o cliente. O Pagar.me[2] e a Zoop[3] fornecem esse tipo de serviço por meio de suas APIs de pagamento.

2.9.1 Zoop

A Zoop é uma empresa que fornece serviços financeiros, permitindo que empresas de qualquer segmento possam operar como instituições de pagamento sem ter que lidar com regulamentações de sistemas financeiros. Ou seja, ela permite que outras organizações tenham acesso a um software pronto, já com todas as regulamentações que o mercado de meios de pagamentos exige, evitando um grande gasto com desenvolvimento e adequação às regras dos meios de pagamentos por parte do cliente (ZOOP, 2020).

2.9.2 Pagar.me

O Pagar.me é uma empresa que fornece serviços de pagamento online, oferecendo um serviço que faz a intermediação de pagamentos entre cliente, vendedor e adquirentes. Isso dá ao adquirente uma solução pronta, atendendo todas as regulamentações que o mercado exige, assim como a Zoop (FINTECH, 2019).

2.9.3 CallBack

A *callback* é uma função que é utilizada quando é necessário responder uma solicitação com várias mensagens ou quando o processamento da mensagem enviada requer uma quantidade grande de tempo (ERL, 2008). Como transações pix e boletos bancários requerem uma certa quantidade de tempo para que o pagamento seja aprovado, a Zoop e Pagar.me utilizam desse mecanismo para notificar a troca de status dessas transações. Logo, no ato de criação de transações deve ser informado uma url *callback* para receber as notificações dos status dos pagamentos.

[2] <https://pagar.me/>
[3] <https://zoop.com.br/>

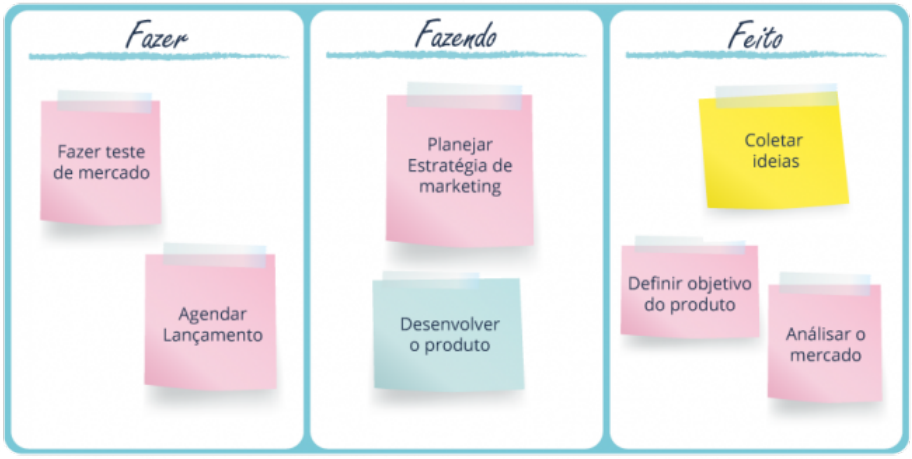
2.10 Objetivos e Resultados-Chave (OKRs)

OKR ou *Objectives and key Results* (Objetivos e Resultados-chave) é uma metodologia de gestão que tem como objetivo organizar, agilizar, controlar e elevar a produtividade da empresa aderente(LACRUZ et al., 2022). Este sistema foi criado em 1970 por Andy Groove, que era o presidente da Intel na época. Posteriormente, diversas empresas do Vale do Silício o adotaram como parte das suas estratégias, como, por exemplo, o Google, LinkedIn, Twitter e GoPro. Como o próprio nome já descreve, essa metodologia é baseada em objetivos e resultados-chave. O primeiro deve ter um prazo para ser concluído e um responsável por ele, geralmente é o gestor. Além disso, a descrição deve ser de uma forma curta e que encoraje e motive os colaboradores a concluí-la. O segundo precisa ser mensurável e quantitativo, é os resultados que serão observados para concluir que o objetivo foi alcançado. A Raises Sistemas utiliza essa metodologia de gestão em todos os seus processos, no processo de desenvolvimento ela era utilizada para definir o prazo da tarefa, como seria avaliada e o que deveria ser construído.

2.11 Kanban

Kanban é um sistema de desenvolvimento japonês criado pela Toyota na década de 60, que é utilizado baseando- se em três componentes, cartões, quadro e colunas. Os cartos mostram o que deve ser construído, a atividade que deve ser desenvolvida. O Quadro serve para organizar todo o processo de desenvolvimento sendo composto pelos cartões e colunas. Já as colunas mostram o estado em que o cartão está. Ele basicamente divide as tarefas que serão realizadas em cartões e utiliza as colunas para referenciar o andamento da atividade. A Figura 2.9 abaixo ilustra um exemplo de como pode ser divido os cartões e colunas.

Figura 2.9 – Exemplo de quadro kanban



Fonte: (ESPINHA, 2019)

3 PROCESSO DE DESENVOLVIMENTO

A definição das tarefas era realizada no início de cada trimestre por meio de uma reunião em que eram apresentados as OKRs do trimestre em questão, possuindo os objetivos pré-determinados e os resultados-chaves. Este último, era utilizado como base para mensurar a progressão da OKR durante o seu desenvolvimento em porcentagem. Nessa reunião, o estagiário era responsável por discutir, entender e sanar todas as dúvidas que ainda possuía sobre o que deveria ser construído. Com todos os objetivos traçados e os resultados-chaves, as atividades eram divididas em cartões no Basecamp[1] para a organização do Kanban do projeto com o uso das colunas *BackLog*, *in Progress* e *Completed*.

No início do desenvolvimento de cada microsserviço, as tarefas eram todas colocadas no *BackLog* e era criado um repositório no GitHub referente ao microsserviço a ser desenvolvido. Nesse repositório, as implementações eram realizadas na *branch* de desenvolvimento. Posteriormente, as tarefas eram atribuídas aos programadores e movida para a coluna *in Progress* durante o desenvolvimento da mesma, onde todas as regras de negócios eram implementadas e após a conclusão da tarefa, ela era movida para a coluna *Completed*. No final de cada OKR, havia uma reunião para discutir as dificuldades encontradas e os objetivos que foram alcançados.

Durante o processo de desenvolvimento, o *tech lead* supervisionava o estagiário passando diversos conhecimentos em relação a melhoria de código e regras de negócios, e a cada 15 dias ocorria uma reunião com a equipe de desenvolvimento para analisar a progressão dos resultados-chaves e discutir as dificuldades encontradas. Além disso, ao longo do desenvolvimento dos microsserviços, foram seguidos dois padrões:

- Primeiro, todas as rotas dos microsserviços foram protegidas com o JSON Web Token (JWT), isso assegura que apenas serviços com as credenciais corretas tenham acesso a essas rotas criadas. Apenas a que recebia o evento vindo da Zoop ou Pagar.me que não era protegida com o JWT, pois eles enviavam o seu próprio *token* que posteriormente era validado em um *endpoint* disponibilizado pela mesma. O *token* era gerado pela API principal com um tempo de expiração pré-definido. Assim, após essa expiração, o *token* já não era mais válido para ser usado.

- Segundo, os dados recebidos e enviados eram registrados no LogFlare para ter um controle maior de possíveis erros no fluxo dos microsserviços. Essas informações eram muito

[1] <https://basecamp.com/>

importantes, pois eram utilizadas para identificar problemas de pagamento reportado por clientes no suporte. Isso permite uma rápida identificação do problema em produção e posteriormente, a sua resolução de forma rápida e prática.

Após a finalização das implementações, o sistema era enviado para o ambiente de homologação utilizando o CloudFlare Worker. Para isso, o *tech lead* passava as chaves secretas necessárias para realizar o *deploy* do serviço no ambiente de desenvolvimento, e essas variáveis eram inseridas no arquivo de configuração do Cloudflare Worker na raiz do projeto. Em seguida, o *deploy* era realizado executando o comando *wrangler publish* pelo terminal do sistema operacional. Após isso, o serviço já estava *online* para ser utilizado e testado pelos próprios desenvolvedores e o *tech lead*. Finalmente, com tudo validado no ambiente de desenvolvimento, o sistema parte para a etapa final: a de produção, que todo o processo era realizada pelo *tech lead*.

4 ATIVIDADES EXECUTADAS

Neste capítulo, serão descritas as atividades realizadas pelo estagiário na Raise Sistemas. Será descrito como foi o treinamento e construído os dois microsserviços de pagamento. O primeiro microsserviço, foi construído para realizar as transações de pagamentos realizadas no sistema da Atos6, que até então era realizada utilizando a API do Pagar.me. O segundo, foi construído como uma nova opção de pagamento para a plataforma do E-Inscrição.

4.1 Treinamento

A primeira etapa foi a de treinamento, durante um período de um mês o estagiário ficou responsável por estudar conceitos básicos e recursos das linguagens JavaScript e TypeScript. Esse estudo foi por meio de pesquisas, leitura de documentações e por um curso online na plataforma digital Udemy. Durante o treinamento, aconteceram diversas capacitações realizadas pelo *tech lead* da equipe, sempre focadas em recursos que a linguagem pode oferecer. O tempo dado para o aprendizado da linguagem foi suficiente, pois o estagiário já tinha um conhecimento prévio básico da linguagem. Contudo, houve um aprendizado ainda maior sobre os recursos que a linguagem oferece, permitindo o desenvolvimento de códigos mais limpos durante o desenvolvimento dos microsserviços.

4.2 Primeiro Microsserviço

O primeiro microsserviço tinha a responsabilidade de criar transações de pagamentos via cartão de crédito e gerar boletos bancários utilizando a API da Zoop.

4.2.1 Desenvolvimento

Logo após o treinamento foi iniciado o desenvolvimento do primeiro microsserviço, sendo construído por três desenvolvedores *back-end*, incluindo o estagiário, e o *tech lead* que supervisionava o que estava sendo construído. A hospedagem do microsserviço foi feita no Cloudflare Workers por permitir o desenvolvimento com ênfase apenas no código, com todo o suporte de segurança gerenciado pela mesma. A linguagem utilizada foi o TypeScript por conta da tipagem que ela possui, trazendo uma maior segurança nos códigos desenvolvidos.

Nas primeiras semanas, o estagiário ficou encarregado de entender as tecnologias que seriam utilizadas para a construção do sistema de pagamento. Primeiramente, foi necessário

estudar a utilização do Cloudflare Worker. Posteriormente, começaram os estudos com foco na documentação da Zoop, que em seguida seria integrada com o microsserviço construído. Após essas etapas, iniciou-se a implementação do microsserviço. No primeiro momento, as regras de negócio foram implementadas com dados estáticos, servindo para simular o que a api da Zoop retornaria, isso permitiu que a aplicação fosse testada isoladamente antes mesmo de estar integrada com a Zoop.

4.2.2 Endpoints

Para esse microsserviço foram construídos quatro *endpoints*, como representado na Figura 4.1. Sendo eles:

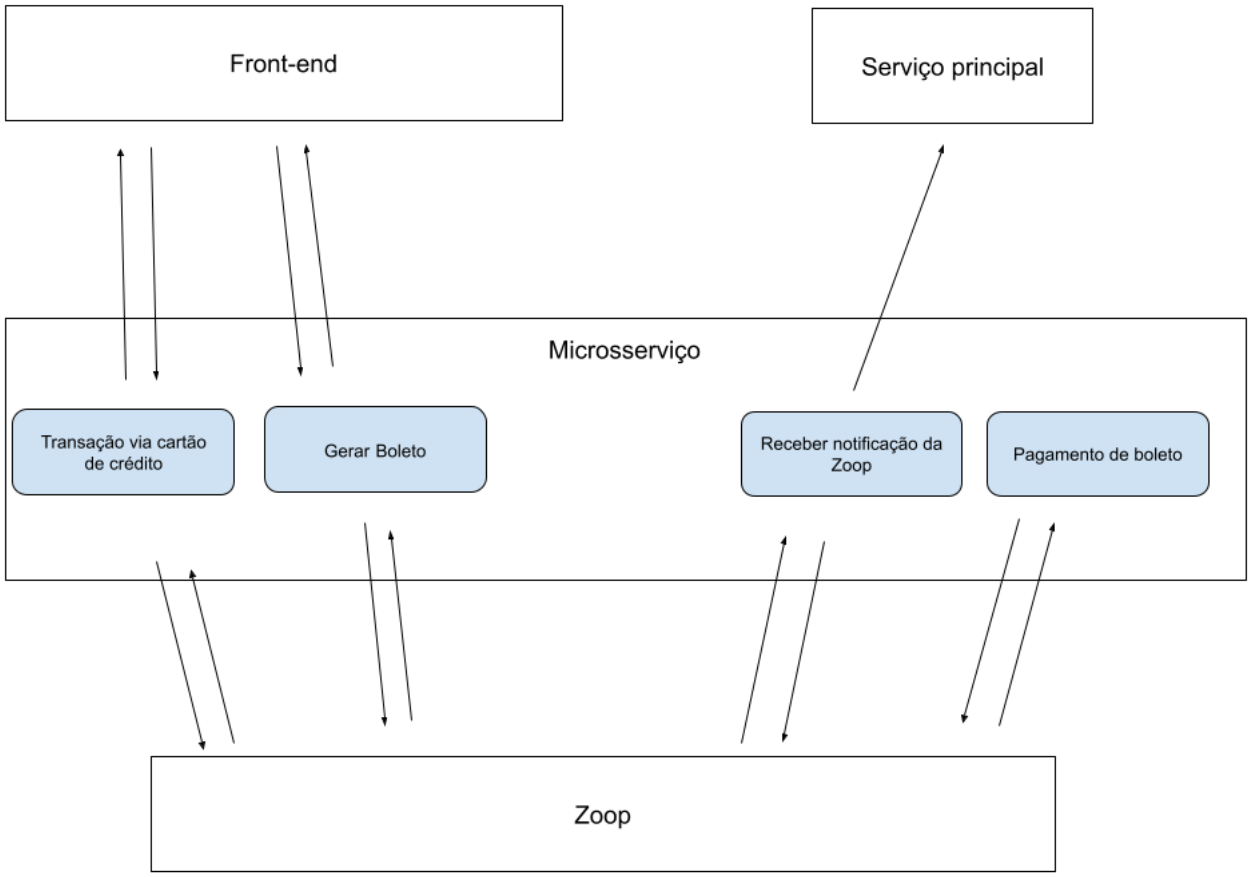
- Transação via cartão de crédito: Construído para realizar o fluxo de transação via cartão de crédito, sendo responsável por receber os dados enviados pelo *front-end*, fazer a comunicação com a Zoop, receber o retorno da comunicação e retornar o status da transação, indicando se ela foi ou não realizada com sucesso. Em caso de falha na transação, uma mensagem indicando o motivo da recusa.

- Gerar Boleto: Responsável por realizar a criação de um boleto. Ele tem a responsabilidade de receber os dados vindos do *front-end*, fazer a comunicação com a Zoop, e com os dados retornados por ela, realizar a montagem do boleto. Por fim, retorna o boleto em pdf.

- Receber notificação da Zoop: Responsável por receber o evento de pagamento de boleto vindo da Zoop. Quando um boleto gerado na Zoop era pago por um cliente, é necessário indicar para o sistema principal da Atos6 que o seu pagamento foi efetivado. Para isso, foi preciso que a aplicação desenvolvida tenha um *endpoint* para receber esse evento. Logo, esse *endpoint* é responsável por receber o evento da Zoop, e de acordo com o evento recebido, fazer a requisição na Zoop, receber a resposta e enviar para o serviço principal da Atos6.

- Pagamento de boleto: É um *endpoint* para teste, com ela é possível realizar pagamento de boleto em ambiente de homologação. Esse *endpoint* é necessário para testar o recebimento da notificação de pagamento de boleto da Zoop. Logo, a função dele é fazer uma requisição na Zoop solicitando a aprovação de um boleto e receber a confirmação da solicitação.

Figura 4.1 – Exemplo do primeiro microsserviço



Fonte: elaborada pelo autor

4.3 Segundo Microsserviço

O segundo microsserivço tinha a responsabilidade de realizar transações de pagamentos via pix para o sistema do E-Inscrição, trazendo uma maior rapidez e facilidade nas transações realizadas por esse sistema. Ele foi iniciado após a conclusão do primeiro microsserviço.

4.3.1 Desenvolvimento

Com um grau de maturidade maior no desenvolvimento de microsserviços por parte do estagiário, ele ficou encarregado por desenvolver esse segundo microsserviço sozinho sendo acompanhado apenas pelo *tech lead* do time. Para este serviço, foi utilizado as mesmas tecnologias do serviço anterior, mudando apenas o gateway de pagamento, que nesse serviço foi utilizada a do Pagar.me. Ele foi escolhido por ter uma taxa menor que a Zoop para esse tipo de transação. Nas primeiras semanas o estagiário ficou delegado a estudar a documentação do

Pagar.me. Posteriormente, foi iniciada a implementação dessa nova demanda. Os acessos ao Pagar.me inicialmente foram construídos todos com retornos estáticos e posteriormente incluídos de fato no código desenvolvido.

4.3.2 Endpoints

Para esse microsserviço foram construídos quatro *endpoints*, como é ilustrado na Figura 4.2. Sendo eles:

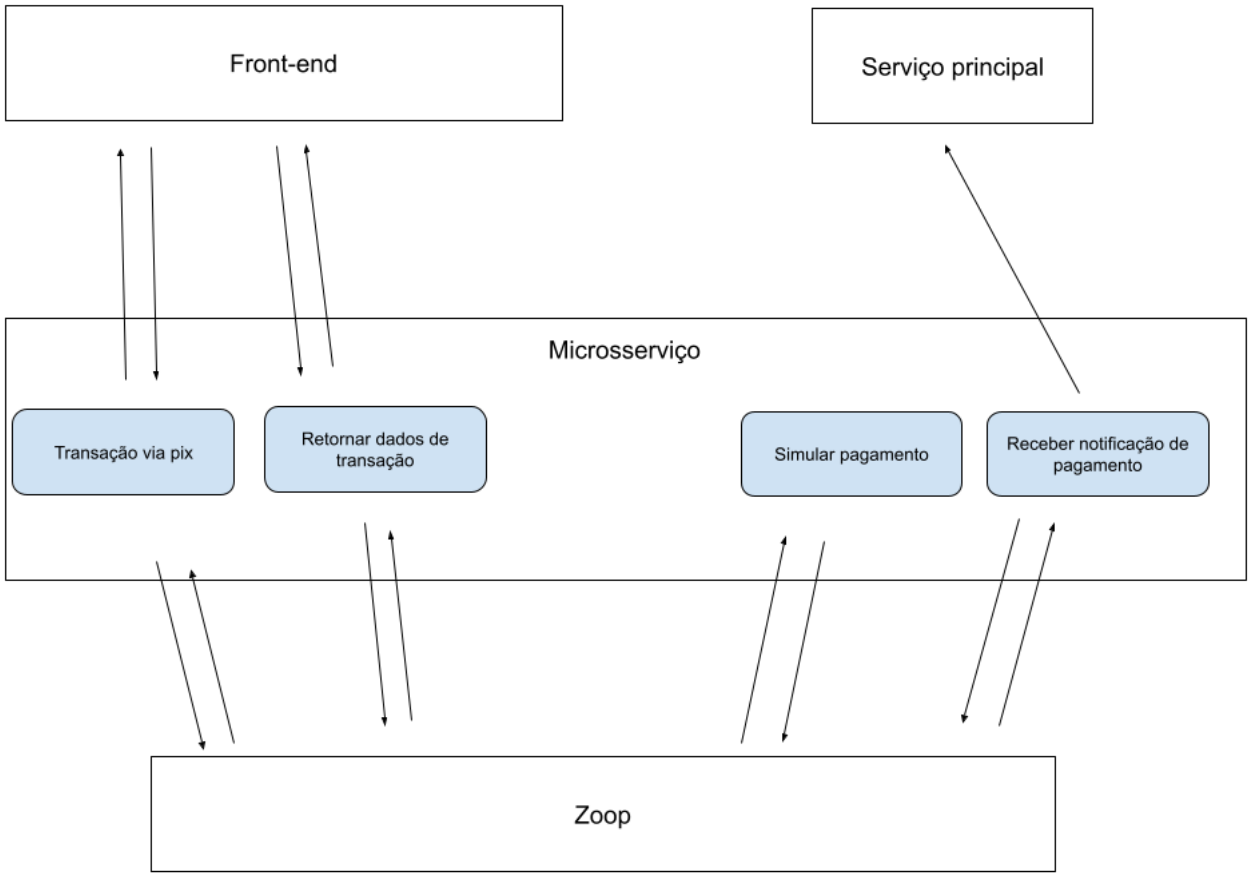
- Transações via pix: Responsável por criar a transação via pix. Para criar uma transação via pix foi necessário realizar a comunicação com o Pagar.me, sendo este o responsável por criar o pix e retornar com todos os dados realcionados ao pix. Logo, ele tinha a responsabilidade de receber a solicitação do *front-end*, fazer a requisição no Pagar.me passando a url *callback*, receber a resposta e retornar os dados para o *front-end*.

- Retornar dados da transação: Como o pagamento via pix é feito de forma instantânea, o *front-end* precisava ter a confirmação do pagamento efetuado pelo cliente, a solução encontrada foi criar esse *endpoint* que tem a responsabilidade de retornar o status de uma transação via Pix buscada, sendo chamada cada 10 segundos para verificar o status do pagamento. Para isso, o *front-end* realizava a solicitação, o endpoint solicitava os dados referentes ao pix buscado, recebia a resposta e enviava para o *front-end* novamente, a cada 10 segundos o fluxo era percorrido novamente.

- Simular pagamento: Responsável por realizar o pagamento de um pix em ambiente de desenvolvimento. Esse endpoint foi criado para que a equipe do *front-end* pudesse testar o fluxo entre as telas do pix com a de confirmação de pagamento. Logo, esse *endpoint* tem a responsabilidade de solicitar o pagamento de um pix gerado e receber a resposta da requisição feita.

- Receber notificação de pagamento: Responsável por receber o evento de pagamento de boleto vindo da Pagar.me. Com o pagamento de um pix, o Pagar.me precisa enviar uma notificação de pagamento, indicando a troca de status da transação. Para isso, esse *endpoint*, realiza a sinalização de que o pagamento foi feito para o serviço principal do E-Inscrição. Isso foi uma solução para garantir que mesmo o cliente pagando o pix sem estar utilizando a tela do *front-end*, a troca de status da transação seria processada pelo serviço principal do E-Inscrição.

Figura 4.2 – Exemplo do segundo microsserviço



Fonte: elaborada pelo autor

5 CONCLUSÃO

Durante o estágio o trabalho em equipe sempre esteve presente, isso permitiu o aprimoramento da habilidade em comunicação e o pensamento coletivo. Além disso, as tomadas de decisões durante o desenvolvimento permitiu ao estagiário ter uma maior percepção na resolução de problemas. Outro aprendizado foi com as tecnologias que foram utilizadas durante o estágio como, JavaScript, TypeScript e versionamento de código com o GIT. O estágio permitiu também uma maior percepção dos fluxos de desenvolvimento de uma empresa de software e como as demandas chegam para o desenvolvedor.

Algumas disciplinas do curso de Ciência da Computação foram de grande importância para o crescimento pessoal durante todo o processo de estágio como, Introdução aos Algoritmos, Estruturas de Dados, Práticas de Programação Orientada a Objetos, Linguagens Formais e Autômatos, e Engenharia de Software. Por meio delas, o estagiário aprendeu o conceito de orientação a objeto, expressões regulares, condicionais, laços de repetições e arquitetura de softwares. A utilização desses conceitos na prática permitiu uma maior percepção da importância do que foi ensinado durante a graduação e o quanto isso é essencial no desenvolvimento de software em geral.

No estágio, o TypeScript foi um grande desafio, por ter experiência apenas com JavaScript o uso de tipagem no primeiro momento foi bem desafiador, erros de sintaxe que o compilador evidenciava era constante, porém ao longo do desenvolvimento isso se tornou muito mais prático e simples, além de tornar o código mais limpo e legível. O uso do Cloudflare Worker para a hospedagem do microsserviço em homologação, instalação e as configurações necessárias para o uso do cloudflare worker no repositório também foi algo muito desafiador, porém com estudos e nivelamento de conhecimento de toda a equipe permitiu uma curva de aprendizado maior de como utilizá-las. Todo o conhecimento adquirido durante o processo de estágio será de grande ajuda para projetos futuros e desenvolvimento pessoal do estagiário.

REFERÊNCIAS

AUTH0. **Introduction to JSON Web Tokens**. [S.l.], 2013. Disponível em: <https://jwt.io/introduction>.

BRASIL, B. C. do. **Estatísticas do pix**. [S.l.], 2022. Disponível em: <https://www.bcb.gov.br/estabilidadefinanceira/estatisticaspix>.

CHACON, B. S. S. **Pro Git**. 2. ed. [S.l.]: Apress, 2014.

CLOUDFLARE. **How Workers works**. [S.l.], 2022. Disponível em: <https://developers.cloudflare.com/workers/learning/how-workers-works/>.

CLOUDFLARE. **O que é uma CDN?** [S.l.], 2022. Disponível em: <https://www.cloudflare.com/pt-br/learning/cdn/what-is-a-cdn/>.

DAYA, S. et al. **Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach**. 1. ed. [S.l.: s.n.].

ERL, T. **SOA Design Patterns**. 1. ed. [S.l.]: Prentice Hall, 2008.

ESPINHA, R. G. **Kanban**. [S.l.], 2019. Disponível em: <https://artia.com/kanban/>.

FENTON, S. **Pro TypeScript**. 2. ed. [S.l.]: Apress Berkeley, 2018.

FINTECH. **O que é Pagar.me**. [S.l.], 2019. Disponível em: <https://fintech.com.br/blog/fintech/o-que-e-pagar-me/>.

FLANAGAN, D. **JavaScript**. 2. ed. [S.l.]: Bookman, 2013.

FOWLER, S. J. **Microsserviços Prontos para Produção**. 1. ed. São Paulo: Novatec, 2017.

LACRUZ, A. J. et al. **Ferramentas de Gestão para Negócios de Impacto Social**. Vitória: Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, 2022.

MALAV, B. **Microservices vs Monolithic architecture**. [S.l.], 2017. Disponível em: <https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4>.

MAROTEL, A. **LOG ANALYSIS**. [S.l.], 2021. Disponível em: <https://www.twaino.com/outils/log-analysis-logflare>.

ROVEDA, U. **O QUE É GIT E GITHUB, COMO USAR E QUAIS SÃO AS VANTAGENS?** [S.l.], 2021. Disponível em: <https://kenzie.com.br/blog/o-que-e-git/>.

SERVICES, A. W. **O que é uma API?** [S.l.], 2022. Disponível em: <https://aws.amazon.com/pt/what-is/api/>.

VINDI, R. **Gateway de pagamento**. [S.l.], 2017. Disponível em: <https://blog.vindi.com.br/gateway-de-pagamento/>.

WOODS, D. **Building Microservices with Spring Boot**. [S.l.], 2015. Disponível em: <https://www.redbooks.ibm.com/redbooks/pdfs/sg248275.pdf>.

ZOOP, R. **O que é a Zoop? Conheça tudo sobre a fintech líder em tecnologia para serviços financeiros**. [S.l.], 2020. Disponível em: <https://zoop.com.br/blog/zoop/servicos-financeiros-zoop-o-que-e-como-funciona-2/>.