RAFAEL DE OLIVEIRA FERNANDES

ARQUITETURA DE MICROSSERVIÇOS COM ÊNFASE EM SISTEMAS DE TRANSAÇÕES

LAVRAS – MG

2022

RAFAEL DE OLIVEIRA FERNANDES

ARQUITETURA DE MICROSSERVIÇOS COM ÊNFASE EM SISTEMAS DE TRANSAÇÕES

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

Prof. Dr. Maurício Ronny de Almeida Souza

Orientador

LAVRAS – MG

2022

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Fernandes, Rafael de Oliveira

Arquitetura de microsserviços com ênfase em sistemas de transações / . 2^a ed. rev., atual. e ampl. – Lavras : UFLA, 2022.

38 p. : il.

Tcc (graduação)–Universidade Federal de Lavras, 2022.

Orientador: Prof. Dr. Maurício Ronny de Almeida Souza.

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

RAFAEL DE OLIVEIRA FERNANDES

ARQUITETURA DE MICROSSERVIÇOS COM ÊNFASE EM SISTEMAS DE TRANSAÇÕES

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

APROVADA em 27 de Abril de 2022.

Prof. Dr. Maurício Ronny de Almeida Souza	UFLA
Prof. Dr. Bruno de Abreu Silva	UFLA
Profa. Dr. Renata Teles Moreira	UFLA

Prof. Dr. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG
2022

Dedico a todos de minha família, com ênfase em minha mãe Alessandra Carla de Oliveira Fernandes e ao meu pai David de Paula Fernandes.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus por todo suporte que me foi dado por ele, afinal sem ele eu nunca teria forças e nem sabedoria para chegar onde cheguei. Gostaria de agradecer aos meus pais por sempre me apoiarem em minhas escolhas, me dando o suporte necessário para seguir em frente junto da sabedoria que tinham a me oferecer. Gostaria de agradecer a toda minha família pelo incentivo fornecido. Gostaria de agradecer também a minha namorada por todo o apoio emocional que a mesma me proporcionou ao longo dessa jornada. Agradeço a todos os meus amigos pelos momentos inesquecíveis durante este curso. Agradeço a Dr. Roseline Rodrigues Moreira Ribas por me ajudar no início de minha jornada, sem ela eu nunca teria conseguido chegar onde cheguei. Agradeço ao Dr. Maurício por todo conhecimento e orientação oferecidas pelo mesmo, junto do apoio durante e após o estágio. Agradeço a cada funcionário da UFLA pelas suas contribuições em minha vida e por fazerem parte dessa etapa em minha vida que ficará marcada para sempre.

O insucesso é apenas uma oportunidade para recomeçar com mais inteligência.
(Henry Ford)

RESUMO

Devido ao cenário de pandemia, diversas empresas mudaram suas políticas gerando desacordo em relação a visão de mercado, marcando assim o fim de algumas parcerias e início de muitas outras. Entre esses acordos os sistemas deveriam escalar e performar de uma maneira mais interessante pelo aumento do fluxo de pessoas utilizando o serviço. Com esse novo cenário, tornou-se necessária uma migração se tratando de *gateway* de pagamento visando suprir as necessidades de um novo tipo de arquitetura de transações que estava sendo implementado pela outra parte da empresa E-inscrição responsável pelo sistema dependente de transações no *gateway* de pagamentos. A partir disso, se mostrou pertinente a implementação de um sistema utilizando a arquitetura de microsserviços para melhor manutenção, escalabilidade e agilidade no desenvolvimento. Esta deveria ser capaz de interligar todos os serviços necessários gerando transações, tratando casos de erros e estornando se necessário, mapeando todas as entradas e saídas a fim de tornar abstrata essa migração para os serviços que utilizam a mesma e por fim atualizando e disponibilizando informações sempre que o status das transações fossem alterados. O projeto foi desenvolvido pelo estagiário como *Back-end* utilizando da arquitetura de microsserviços na linguagem TypeScript, mapeada para JavaScript. Contando com três microsserviços, sendo dois desses utilizando a CDN Cloudflare com o outro microsserviço sendo hospedado pelo Heroku, e outras ferramentas como o Logflare e Wasabi para ajuda na manutenção, análise do sistema em produção e rápido transporte de dados a fim de atualizar o cliente sempre que houvesse mudanças.

Palavras-chave: Gateway de pagamento. Arquitetura de microsserviços. Transações. Back-end. Typescript. Javascript. CDN. Cloudflare. Heroku. Logflare. Wasabi.

ABSTRACT

Due to the pandemic scenario, several companies changed their policies generating disagreement about business vision, thus marking the end of some partnerships and the beginning of many others. Among these agreements, the systems should scale and perform in a more interesting way by increasing the flow of people using the service. With the new scenario, a migration became necessary in the case of payment gateway in order to meet the needs of a new type of transaction architecture that was being implemented by the other part of the company E-inscrição responsible for the system dependent on transactions in the payment gateway. From that, it was relevant to implement a system using the microservices architecture for better maintenance, scalability and agility in development. It should be able to link all the necessary services, generating transactions, handling cases of errors and reversing if necessary, mapping all inputs and outputs in order to make this migration abstract for the services that use it and finally updating and making information available whenever the status of transactions was changed. The project was developed by the intern as a Back-end using the microservices architecture in the TypeScript language, mapped to JavaScript. Counting on three microservices, two of which using CDN Cloudflare with the other microservice being hosted by Heroku, and other tools like Logflare and Wasabi to help with maintenance, analysis of the system in production and fast data transport in order to update the client whenever there were changes.

Keywords: Payment gateway. Microservices architecture. Transaction. Back-end. Typescript. Javascript. CDN. Cloudflare. Heroku. Logflare. Wasabi.

LISTA DE FIGURAS

Figura 2.1 – Fluxo do Node.js 16
Figura 2.2 – Controle de aprovação e reprovação de alunos em JavaScript 17
Figura 2.3 – Exemplo de tipagem do objeto *pessoa* utilizando TypeScript 18
Figura 2.4 – Exemplo do formato JSON 19
Figura 2.5 – Lógica da arquitetura de microsserviços 20
Figura 2.6 – Exemplo de uma simples implementação para o formato aceito pelo Cloudflare . 22
Figura 2.7 – Exemplo de fluxo utilizando BullMQ 23
Figura 2.8 – Exemplo do fluxo apresentado pelo gateway de pagamento no projeto 25
Figura 3.1 – Figura exemplificando o fluxo da transação sem considerar erros 29

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 Introdução 12
1.1 Sobre 'E-inscrição' 12
1.2 Organização do Trabalho 13
2 Conceitos e Tecnologias 14
2.1 Desenvolvimento Web 14
2.1.1 Node.js 15
2.1.2 JavaScript 16
2.1.3 JavaScript Object Notation (JSON) 18
2.2 Arquitetura de microsserviços 19
2.2.1 Content Delivery Network (CDN) 20
2.2.2 Heroku 22
2.2.3 BullMQ 23
2.3 Outras Tecnologias 24
2.3.1 Gateway de Pagamento 24
2.3.2 Git/Github 25
3 Atividades Desenvolvidas 26
3.1 Necessidades do projeto 26
3.2 Visão geral da Arquitetura 27
3.3 Microsserviço de transação 30
3.3.1 Geração de Transações 30
3.3.2 Geração de inscrição 31
3.3.3 Armazenamento de status da transação 31
3.4 Microsserviço de enfileiramento 31
3.5 Microsserviço de validação 32
3.6 Observação e manutenção 32
3.7 Considerações Finais 33
4 Conclusão 35

REFERÊNCIAS 37

1 INTRODUÇÃO

A empresa E-inscrição [1] fornece serviços de gestão de eventos com uma série de ferramentas disponibilizadas por eles. Com isso o controle de transações para liberação de inscrições é de extrema importância, tornando assim necessário a presença de *gateways* de pagamento intervindo nas formas como ocorrem estas transações.

Após mudanças burocráticas em relação ao *gateway* de pagamentos utilizado, o mesmo se mostrou incoerente com os propósitos e diretrizes da empresa, revelando a necessidade assim em recorrer a um *gateway* mais adaptativo, de acordo com as expectativas da empresa. Tendo em vista tal cenário, este relatório de estágio tem como finalidade relatar o desenvolvimento de um sistema baseado em arquitetura de microsserviços para o processamento de transações de pagamento na empresa E-inscrição, especificando o cenário e motivo do projeto em si.

O estágio teve duração de 12 meses, durante o período de fevereiro de 2021 até fevereiro de 2022, contando com uma equipe composta por três desenvolvedores *Back-ends* e três *Front-ends* junto com um *tech Lead*. O estagiário participou da elaboração do problema, controle de risco em relação a partes sensíveis do sistema, escolha de arquitetura, definição de microsserviços a serem desenvolvidos, desenvolvimento dos microsserviços e testes para detecção de erros críticos no sistema produzido. Este relatório descreve as atividades desenvolvidas pelo estagiário, desafios encontrados, metodologias utilizadas e decisões tomadas para adaptar o projeto de acordo com a demanda da empresa e demonstrar o amadurecimento profissional.

1.1 Sobre 'E-inscrição'

A empresa E-inscrição é especializada em gestão de eventos no mercado religioso, se responsabilizando por todo o processo de inscrição nos eventos dos seus respectivos organizadores e lidando com a parte de pagamento dessas inscrições de maneira abstrata para quem organiza. Uma das características marcantes da empresa é o fato dessa ser remota desde a sua criação, valorizando a privacidade e qualidade de trabalho com comodidade por parte de seus colaboradores, visando estreitar relacionamentos sempre com ênfase em uma boa comunicação para com aqueles presentes.

[1] https://home.e-inscricao.com/

A empresa trabalha com uma adaptação da metodologia ágil SCRUM (SOARES, 2004), seguindo também o modelo de OKR's, uma fórmula para definir metas, em forma trimestral a fim de otimizar os processos implementados nesta. A empresa preza pelas entregas propostas mas intensifica seu olhar sobre a qualidade de tudo que é desenvolvido visando segurança e confiabilidade, se mantendo em constante contato com seus colaboradores e clientes para garantir isto. Esta se mantém aberta a ideias para inovação sempre que possível atendendo às necessidades do mercado respeitando a singularidade de cada um de seus clientes.

A empresa é dividida em diversos setores, entre estes o estagiário participou ativamente no setor de *tech*, responsável por toda a parte de desenvolvimento de produtos, como desenvolvedor back-end trabalhando diretamente em conjunto com o setor de *Customer Success* (CS), que tem como objetivo repassar e resolver o mais rápido possível as insatisfações dos clientes em relação ao sistema. O CEO e o *tech Lead* também estavam em contato direto com o projeto para garantir que a regra de negócio estaria sendo atendida como o indicado já que o estagiário era novo na empresa.

1.2 Organização do Trabalho

Além deste capítulo introdutório, este documento está organizado da seguinte forma. O Capítulo 2 apresenta os conceitos e tecnologias empregados no desenvolvimento das atividades durante o estágio. O Capítulo 3 descreve as atividades desenvolvidas pelo estagiário, com ênfase na descrição da solução desenvolvida e os microsserviços implementados. O Capítulo 4 conclui o trabalho com a visão do estagiário sobre seus principais aprendizados.

2 CONCEITOS E TECNOLOGIAS

Este capítulo apresenta a definição de conceitos e tecnologias utilizadas ao longo do projeto relatado pelo estagiário. Dada à natureza do projeto, os conceitos e tecnologias apresentados estão organizados em Desenvolvimento Web (Seção 2.1), Arquitetura de Microsserviços (Seção 2.2), e outras tecnologias (Seção 2.3). As tecnologias utilizadas no projeto foram escolhidas junto a equipe de desenvolvimento, visando melhores resultados de acordo e fundamentadas nos pilares de: (i) ênfase ao conceito de microsserviços, (ii) agilidade e (iii) praticidade no desenvolvimento e manutenção do serviço proposto.

2.1 Desenvolvimento Web

Desenvolvimento Web se trata de tecnologias gerais voltadas para desenvolvimento de sites, sistemas e quaisquer ferramentas que constituem a consolidada internet. Costuma-se dividir tais tecnologias em duas camadas de desenvolvimento, sendo estas:

- O *Back-end*, responsável pela parte de desenvolvimento que interage com o que está por trás das aplicações em desenvolvimento, abrangendo assim toda a estrutura de apoio do sistema por si só (OLIVEIRA et al., 2019);
- O *Front-end*, responsável pelo desenvolvimento da interface gráfica que está em contato interagindo diretamente com os usuários a fim de captar e disponibilizar informações importantes para estes (OLIVEIRA et al., 2019);

Englobada pelo *Back-end*, a arquitetura REST (*Representational State Transfer*) ou transferência de estado representacional, é um um estilo arquitetural de software. Este foi desenvolvido a fim de servir aplicações web oferecendo diversas linhas de orientação. Com isto, pode-se desenvolver um serviço escalável e de alta performance independente das plataformas utilizadas e linguagens (MARQUES, 2018). Como o nome sugere, este estilo arquitetural permite a comunicação entre aplicações, podendo assim estabelecer uma conexão com o servidor de destino e enviar uma requisição com o endereço buscado. Esta organização é a base de diversos tipos de arquitetura, como por exemplo a arquitetura de microsserviços.

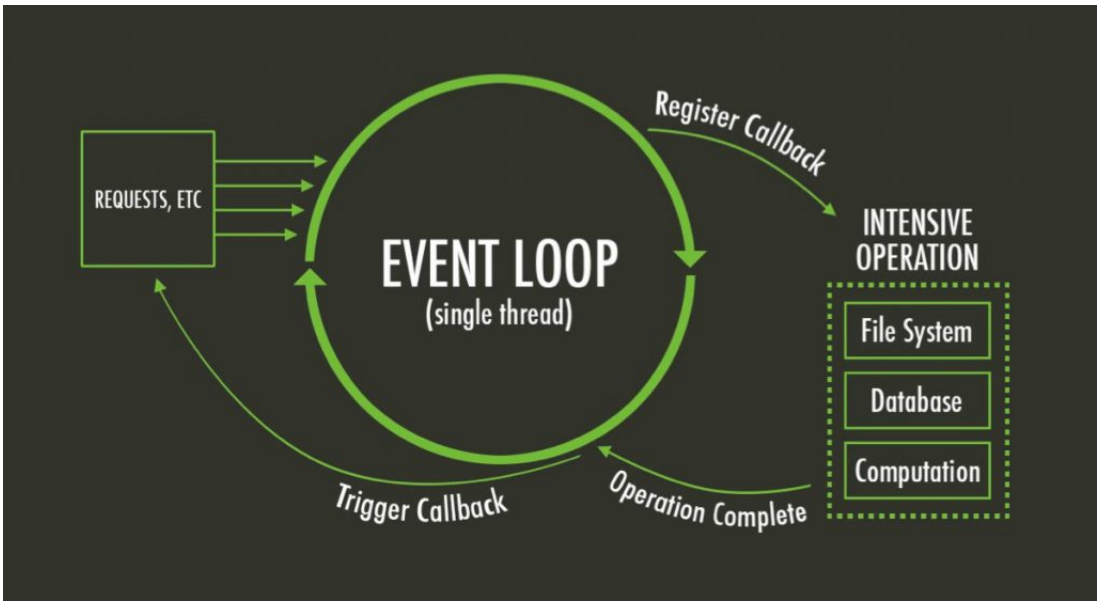
2.1.1 Node.js

O Node.js foi criado em 2009 pelo pesquisador Ryan Dahl e desenvolvido a partir da linguagem JavaScript. Tendo como intuito melhorar o desempenho das aplicações web utilizando apenas códigos em JavaScript. O Node.js complementou o uso da linguagem JavaScript já que permitiu o desenvolvimento tanto por parte do *Front-end* quanto do *Back-end* (BARSOTI; GIBERTONI, 2020).

De acordo com (DOCS, 2021), um dos diferenciais relacionado ao Node.js é o fato de este ter *thread* única, enquanto nos modelos tradicionais existem múltiplas *threads*. Desse modo, as requisições são tratadas como eventos ao invés de gerar novas *threads* acarretando em uma redução nas chances de interrupção por sobrecarga de requisições. Esse conjunto de características do Node.js garante a simplificação dos processos de demanda e evitam diversas interrupções pelo paralelismo, permitindo alta produtividade com sua alta capacidade de escalabilidade, flexibilidade, baixo custo e modernismo se tratando do uso de padrão arquitetural de entrada e saída assíncrona (OLIVEIRA; ZANETTI, 2021).

A figura 2.1 representa o fluxo de uma requisição utilizando a *Event Loop* do Node.js. Em tal imagem, podemos ver que todas as requisições passam por um fluxo de encaminhamento via *thread* única para um registrador de funções *callbacks*. Tais funções tem um comportamento diferente. Estas podem ser definidas como funções que ditam uma instrução após o término de uma operação. Garantindo assim que após sua conclusão, esta possa ser acionada via *Event Loop*, sendo encaminhada para sua conclusão. Tal fluxo permite o processamento apenas quando um evento for acionado, gerando assim menos problemas e maior otimização se relacionado ao processamento em si.

Figura 2.1 – Fluxo do Node.js



Fonte: (SANCHEZ-PALENCIA, 2019)

2.1.2 JavaScript

De acordo com (FLANAGAN, 2004), JavaScript é uma linguagem de programação da Web utilizada amplamente pela maioria dos sites e navegadores modernos. A linguagem foi criada em 1995 com o propósito de ser uma linguagem de *scripts* para páginas Web, mas com o decorrer dos anos se adaptou também para ambientes sem navegadores (mais propriamente dito como Node.js). O JavaScript tem como característica o fato de ser uma linguagem interpretada e baseada em objetos com funções de primeira classe.

Ao longo dos anos esta vem sendo modificada devido ao seu envio para padronização a *European Computer Manufacturers Association (ECMA)*. Com isso, esta teve seu nome modificado para ECMAScript mesmo ainda sendo citada pelo seu nome inicial JavaScript. Com tal padronização, a mesma vem recebendo atualizações com suporte a novas sintaxes visando facilitar o desenvolvimento (FLANAGAN, 2004). Apesar da semelhança no nome e serem mantidas pela mesma empresa, a linguagem JavaScript não foi originada da linguagem Java e tem diferenças consideráveis em relação a esta.

Com todas as praticidades e adaptabilidade, JavaScript se mostra como a linguagem mais utilizada, segundo levantamento realizado pelo Github (2021), sendo assim uma das linguagens mais populares para desenvolvimento.

A Figura 2.2 apresenta um exemplo de código escrito em JavaScript. O código apresenta um exemplo de manipulação de vetores para determinar se um aluno foi aprovado ou não, de acordo com a sua frequência e média de notas.

Figura 2.2 – Controle de aprovação e reprovação de alunos em JavaScript

```
/*
Autor: Rafael de Oliveira Fernandes
Código de exemplo para controle de aprovação ou reprovação de alunos
*/

let aluno1 = {
  nome: "Aluno1Teste",
  notas: [60, 60, 50, 70, 75],
  frequencia: 62,
};
let aluno2 = {
  nome: "Aluno2Teste",
  notas: [30, 40, 50, 60, 70],
  frequencia: 95,
};
let aluno3 = {
  nome: "Aluno3Teste",
  notas: [60, 60, 85, 70, 75],
  frequencia: 50,
};
let aluno4 = {
    nome: "Aluno4Teste",
    notas: [60, 60, 60, 60, 60],
    frequencia: 75,
  };
const alunosInfo = [aluno1, aluno2, aluno3, aluno4];
alunosInfo.forEach((aluno) => {
  let nota = 0;
  aluno.notas.forEach((notas) => {
    nota += notas;
  });
  nota/aluno.notas.length >= 60 && aluno.frequencia >= 75
    ? console.log(
        `O aluno ${aluno.nome} foi aprovado com a média de ${nota/aluno.notas.length} e frequência de ${aluno.frequencia}%`
      )
    : console.log(
        `O aluno ${aluno.nome} foi reprovado com a média de ${nota/aluno.notas.length} e frequência de ${aluno.frequencia}%`
      );
});
```

Fonte: (Autor, 2022)

Uma das inovações baseadas em JavaScript é seu superconjunto (conjunto de ferramentas), chamado TypeScript[1], que vem para complementar com todos os novos recursos implementados pela Microsoft. Um dos complementos é o suporte a utilização de interfaces de forma clara juntamente com a tipagem fornecida pelo TypeScript. É necessário ressaltar que na maioria dos casos todas essas vantagens serão perdidas, pois o código em TypeScript precisa ser mapeado para a linguagem em JavaScript. Apesar dessa perda, o mesmo ainda se mostra interessante por facilitar no desenvolvimento ao prover sintaxes mais recentes e permitir a identificação de mais erros durante a implementação

[1] https://www.typescriptlang.org

dos códigos. Logo, o TypeScript pode ser visto como uma ferramenta que potencializa o JavaScript (MICROSOFT, 2022).

A Figura 2.3 apresenta um trecho de código exemplificando a tipagem de um objeto "pessoa". Isto garante que todas as funções utilizadas serão compatíveis com seus respectivos tipos, evitando conversões acidentais.

Figura 2.3 – Exemplo de tipagem do objeto *pessoa* utilizando TypeScript

```typescript
/*
Autor: Rafael de Oliveira Fernandes
Código de exemplo para tipagem utilizando typescript
*/
type PessoaDto = {
  nome: string;
  idade: number;
  dataDeNascimento: Date;
};

const exemploFuncao = (pessoa: PessoaDto) => {
  console.log(pessoa);
};

const pessoaExemplo: PessoaDto = {
  nome: "Rafael",
  idade: 24,
  dataDeNascimento: new Date("06/04/1998"),
};

exemploFuncao(pessoaExemplo);
```

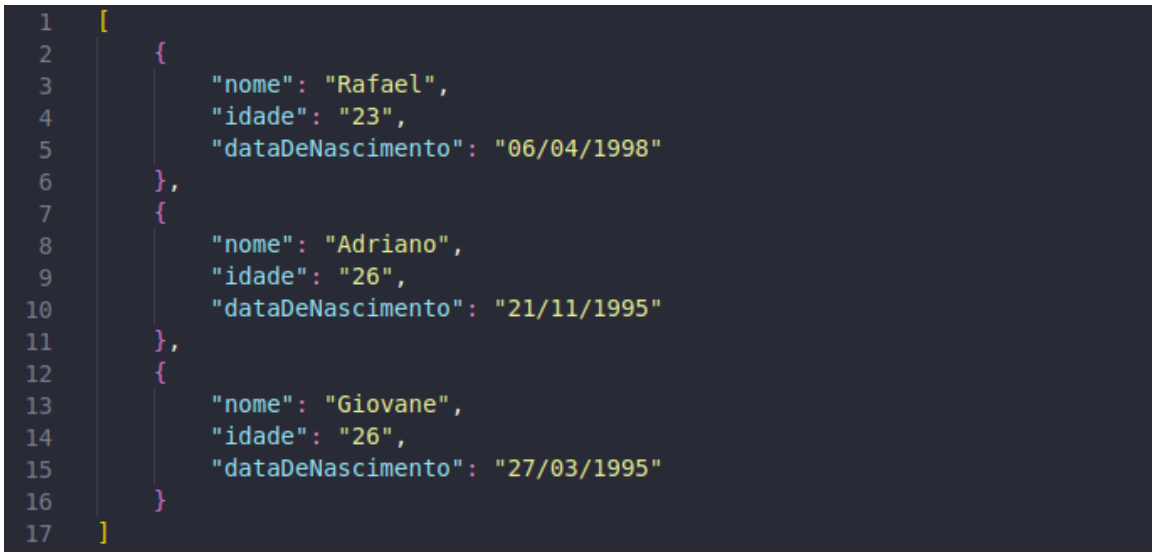
Fonte: (Autor, 2022)

2.1.3 JavaScript Object Notation (JSON)

JavaScript Object Notation foi originado por Douglas Crockford derivada de uma simples representação de objetos e arrays do JavaScript referenciando o lado do cliente (EMER, 2013). Pode-se especificar este como uma notação de objetos, matrizes, número, entre outros, que tem como objetivo a representação de uma forma comum a diversas linguagens.

Uma de suas características é a facilidade que o mesmo é trafegado entre as aplicações em quaisquer protocolos. Este padrão se tornou popular devido a sua simplicidade, substituindo em diversas linguagens o padrão XML como um formato de intercâmbio de dados (FONSECA; SIMÕES, 2007).

Figura 2.4 – Exemplo do formato JSON



```
[
    {
        "nome": "Rafael",
        "idade": "23",
        "dataDeNascimento": "06/04/1998"
    },
    {
        "nome": "Adriano",
        "idade": "26",
        "dataDeNascimento": "21/11/1995"
    },
    {
        "nome": "Giovane",
        "idade": "26",
        "dataDeNascimento": "27/03/1995"
    }
]
```

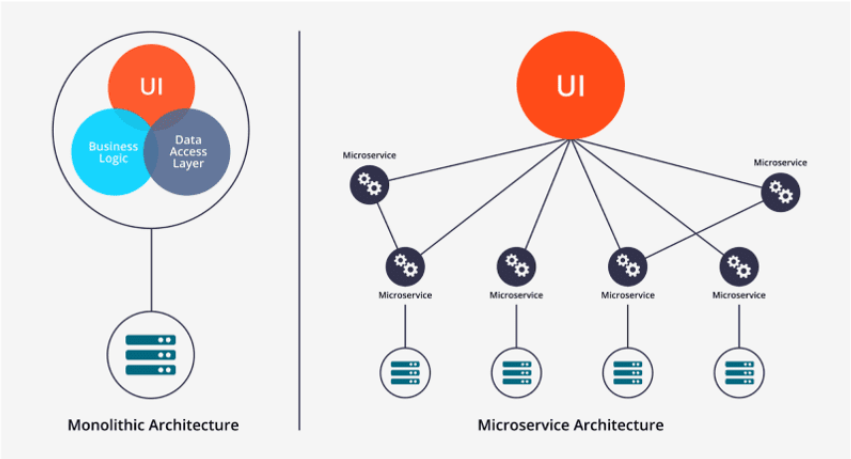
Fonte: (Autor, 2022)

2.2 Arquitetura de microsserviços

Surgindo como uma alternativa a arquitetura monolítica, ao contrário desta onde toda a aplicação se concentra em um único lugar, os microsserviços são um conjunto de pequenos serviços independentes que devem ser implementados com um único propósito limitado pelo contexto e que juntos trabalham para realizar tarefas (MAZLAMI; CITO; LEITNER, 2017). Esse tipo de abordagem traz grandes vantagens como por exemplo a leveza da aplicação, capacidade de compartilhamento de processos semelhantes entre aplicações e principalmente na agilidade de desenvolvimento e entrega do software com qualidade (VILLAÇA; JR; AZEVEDO, 2018).

A Figura 2.5 demonstra o funcionamento do fluxo de dados na arquitetura de microsserviços de forma ideal. Podem-se observar algumas diferenças na implementação da arquitetura de microsserviços em relação a monolítica tradicional.

Figura 2.5 – Lógica da arquitetura de microsserviços



Fonte: (MALAV, 2017)

2.2.1 Content Delivery Network (CDN)

Content delivery Network (CDN) pode ser vista como um conjunto de servidores replicados a partir do principal, distribuídos geograficamente a fim de acelerar a entrega de conteúdos da web, sendo capaz de aproximar a aplicação tendo como referência o usuário (CLOUDFLARE, 2022b).

No contexto de microsserviços, CDN é importante para agilizar e facilitar o desenvolvimento. Garantindo que a única preocupação será com a lógica em si, já que a CDN oferece todas as ferramentas de segurança e hospedagem para este.

A ideia foi desenvolvida com o intuito de resolver problemas relacionados a congestionamento de rede causado por sobrecarga referente a conteúdos considerados de extrema complexidade, como por exemplo um gráfico ou vídeo da internet.

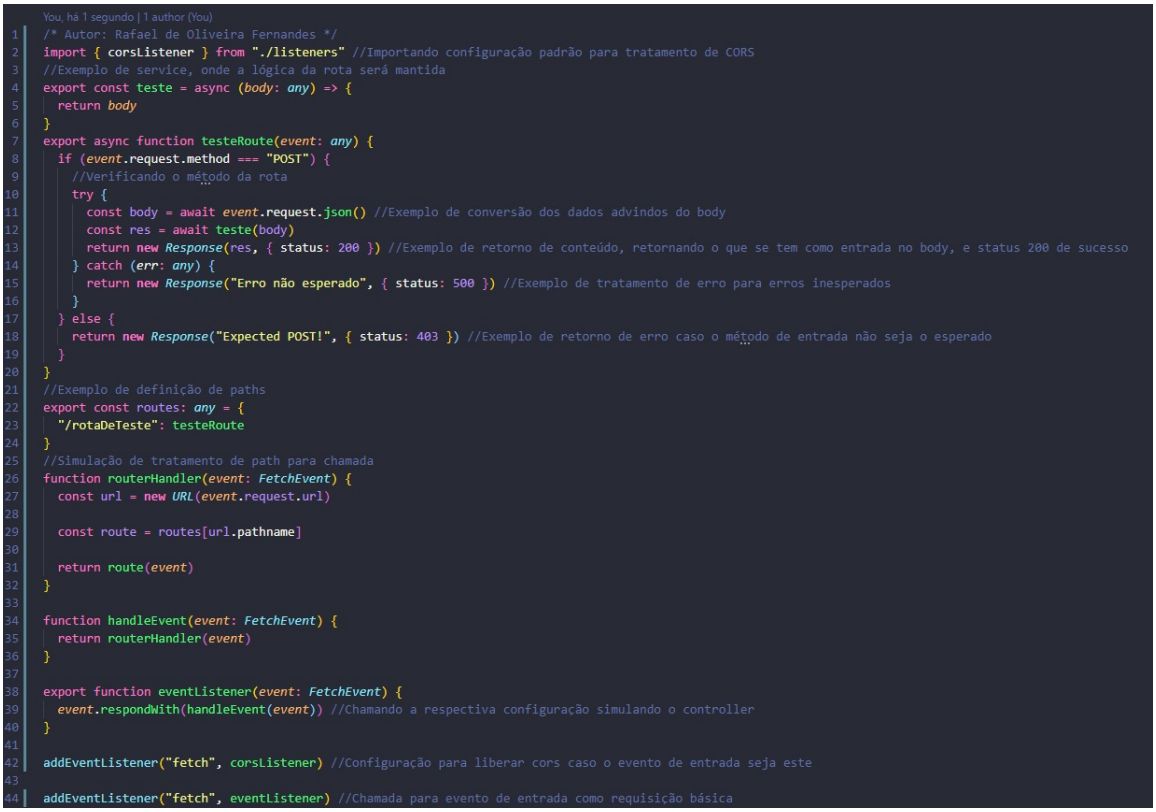
Criada em 2009 por Matthew Prince, Lee Holloway e Michelle Zatlyn, a Cloudflare[2] se mostra como um serviço de CDN que trabalha como um pacote de serviços, fornecendo segurança em relação a proteção de dados e permitindo o armazenamento de serviços na arquitetura de microsserviços, entre outras (CLOUDFLARE, 2022a). A mesma ficou marcada em fevereiro de 2014 após mitigar o maior ataque DDoS registrado até a respectiva data, atingindo 400 Gbit/s (MUSIL, 2014), destacando-a de

[2] https://www.cloudflare.com/pt-br/

outras que vinham a oferecer o mesmo produto. O Cloudflare torna possível a adição de microsserviços na forma de ambientes nomeados "*workers*"que garantem ao desenvolvedor preocupação apenas com o código e a lógica em si, entregando a proteção necessária juntamente com serviços essenciais para este de forma simples.

A Figura 2.6 se trata de um exemplo de estruturação de código para desenvolvimento utilizando Cloudflare. Em tal exemplo, podemos ver que o desenvolvimento é baseado em programação a eventos. Começando no *addEventListener* como a plataforma Node.js garantindo assim uma simplicidade pela similaridade de desenvolvimento. Após a adição da requisição em forma de evento na lista de eventos do Cloudflare, temos a chamada da função *eventListener*, responsável por encaminhar requisições para a função *routerHandler*, responsável por controlar as rotas de acordo com seus respectivos endereços e configurações. Após tal encaminhamento, a lógica e a configuração da rota requisitada é aplicada, gerando uma resposta por meio desse evento que é retornado para quem requisitou ao final do fluxo.

Figura 2.6 – Exemplo de uma simples implementação para o formato aceito pelo Cloudflare



```
You, há 1 segundo | 1 author (You)
/* Autor: Rafael de Oliveira Fernandes */
import { corsListener } from "./listeners" //Importando configuração padrão para tratamento de CORS
//Exemplo de service, onde a lógica da rota será mantida
export const teste = async (body: any) => {
  return body
}
export async function testeRoute(event: any) {
  if (event.request.method === "POST") {
    //Verificando o método da rota
    try {
      const body = await event.request.json() //Exemplo de conversão dos dados advindos do body
      const res = await teste(body)
      return new Response(res, { status: 200 }) //Exemplo de retorno de conteúdo, retornando o que se tem como entrada no body, e status 200 de sucesso
    } catch (err: any) {
      return new Response("Erro não esperado", { status: 500 }) //Exemplo de tratamento de erro para erros inesperados
    }
  } else {
    return new Response("Expected POST!", { status: 403 }) //Exemplo de retorno de erro caso o método de entrada não seja o esperado
  }
}
//Exemplo de definição de paths
export const routes: any = {
  "/rotaDeTeste": testeRoute
}
//Simulação de tratamento de path para chamada
function routerHandler(event: FetchEvent) {
  const url = new URL(event.request.url)

  const route = routes[url.pathname]

  return route(event)
}

function handleEvent(event: FetchEvent) {
  return routerHandler(event)
}

export function eventListener(event: FetchEvent) {
  event.respondWith(handleEvent(event)) //Chamando a respectiva configuração simulando o controller
}

addEventListener("fetch", corsListener) //Configuração para liberar cors caso o evento de entrada seja este

addEventListener("fetch", eventListener) //Chamada para evento de entrada como requisição básica
```

Fonte: (Autor, 2022)

2.2.2 Heroku

Heroku[3] é uma plataforma de serviço baseada em um sistema de contêiner gerenciado, com serviços de dados integrados junto de um poderoso ecossistema, para implantação e execução de aplicativos modernos (HEROKU, 2022). A experiência do desenvolvedor Heroku é uma abordagem centrada em aplicativos para entrega de software, integrada às ferramentas e fluxos de trabalho de desenvolvedor mais populares da atualidade (HEROKU, 2022).

Desenvolvido por Adam WIiggins, Orion Henry e James Lindenbaum, surgiu com o proposito de apresentar suporte a projetos compatíveis com Rack, interface modular de servidor *web* Ruby, sendo desenvolvido em pouco tempo. Diversos problemas foram enfrentados por este devido a falta

[3] https://www.heroku.com/platform

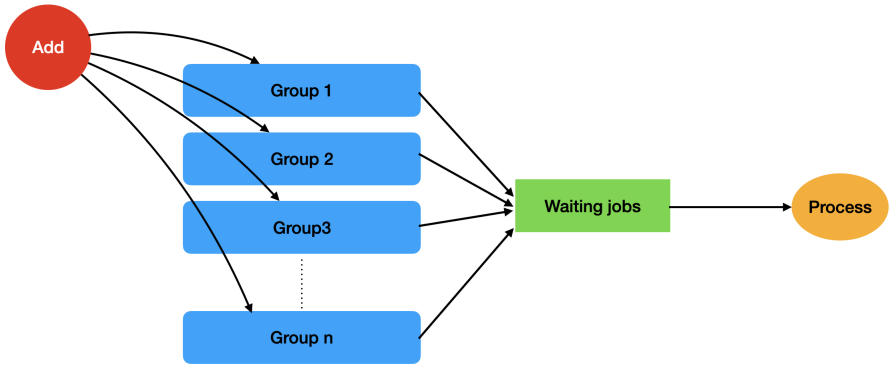
de investimentos por parte dos clientes. Em 2009 a versão renovada do Heroku foi lançada após alguns meses de desenvolvimento, com diversos acréscimos em relação a ferramentas.

2.2.3 BullMQ

BullMQ[4] se trata de uma biblioteca em Node.js que tem como propósito o enfileiramento de requisições utilizando o Redis, armazenador de estrutura de dados de chave-valor com código aberto e na memória, como forma de armazenamento para futuro processamento das mesmas (BULLMQ, 2022b). Tem como uma das principais vantagens uma boa escalabidade horizontal permitindo o redimensionamento através da adição de mais *workers*.

A Figura 2.7 demonstra o fluxo dos dados no enfileiramento pelo BullMQ. Inicialmente a requisição é adicionada e armazenada no Redis para futuro processamento. Aguardando assim na fila para ser trazido de volta e processado por meio dos "trabalhadores". Estes são responsáveis pela lógica ditando o que deve ser feito com os dados da requisição quando for chegada a vez de processar o mesmo.

Figura 2.7 – Exemplo de fluxo utilizando BullMQ



Fonte: (BULLMQ, 2022a)

[4] https://docs.bullmq.io

2.3 Outras Tecnologias

Nesta seção são apresentados outros conceitos e tecnologias relevantes ao projeto desenvolvido pelo estagiário.

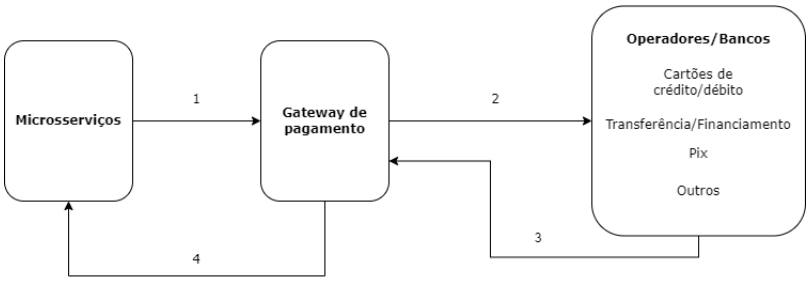
2.3.1 Gateway de Pagamento

Gateway de pagamento pode ser visto como uma tecnologia que engloba pagamentos digitais visando a comunicação entre o consumidor, o banco e as operadoras de cartão de crédito, possibilitando assim conectar diferentes adquirentes ao respectivo negócio e consequentemente centralizar todas as operações de pagamento (PAGAR.ME, 2022).

Os *gateways* de pagamento fornecem um processo muito importante para os desenvolvedores que a utilizam chamado *postback*. Este pode ser visto como um retorno/envio do próprio *gateway* (cliente) mediante a qualquer alteração pertinente em uma transação se tratando do cenário em que se encontra esta (BERTUOL, 2011). Em transações, por exemplo, é muito comum que postbacks sejam utilizados para qualquer alteração no status da mesma como forma de gatilhos. Alertando assim o responsável para que o sistema notificado tome as ações configuradas.

A figura 2.8 representa um pequeno exemplo do fluxo pertinente a criação de uma transação com sucesso no *gateway* de pagamentos, demonstrando como é feita a comunicação do *gateway* com os operadores no projeto relatado. Inicialmente o microsserviço envia a requisição com todos os dados mapeados conforme exigido pelo *gateway* de pagamentos (1). Em seguida o *gateway* se encarrega de comunicar com os operadores/bancos, abstraindo assim qualquer configuração relacionada a pagamentos do próprio microsserviço (2). Após essa comunicação, os operadores/bancos respondem a requisição, informando os dados gerados a partir dessa transação (3). Por fim o *gateway* de pagamento responde ao pedido do microsserviço e envia um postback informando que a transação foi criada (4).

Figura 2.8 – Exemplo do fluxo apresentado pelo gateway de pagamento no projeto



Fonte: (Autor, 2022)

2.3.2 Git/Github

Git[5] se trata de uma ferramenta responsável pelo controle de versão dos projetos em desenvolvimento, se destacando por ser gratuita. Ela permite a criação de um histórico de alterações dos códigos-fonte relacionado aos projetos em desenvolvimento. Garantindo uma organização e controle quando se trata de mudanças constantes em códigos (CUNHA, 2018).

No entanto, a mesma pode ser complementada por diversos sistemas, como é o caso do Github[6]. Este pode ser definido como um repositório remoto responsável por hospedar os repositórios do Git. O mesmo funciona como um sistema responsável por agregar todas as modificações realizadas pela equipe de desenvolvedores, permitindo a unicidade de diferentes versões de código juntamente com seus históricos (MONTEIRO EDUARDA, 2021).

[5] https://git-scm.com
[6] https://github.com

3 ATIVIDADES DESENVOLVIDAS

Neste capítulo, serão descritas as atividades realizadas pelo estagiário no desenvolvimento dos microsserviços que constituem o sistema de transações da empresa E-inscrição.

A gestão do projeto e o controle de versões foi realizado com a ferramenta Github. O projeto foi inicialmente planejado junto com superiores para identificação de necessidades do projeto. Ao longo do desenvolvimento, o estagiário participou de reuniões periódicas com superiores a fim de discutir pontos de falhas críticas e estratégias para prevenção e contenção. Caso necessário, a geração de alertas naqueles que não pudessem ser tratados deveria ser feita.

As atividades do estagiário foram supervisionadas por um desenvolvedor sênior que passou diversos conhecimentos e fez muitas observações sobre melhorias tanto de código quanto de regras de negócio.

O supervisor lembrava, sempre que necessário, os pontos de risco a serem observados de forma solícita até a entrega, garantindo confiabilidade a partir de testes feitos pelo próprio estagiário e pela equipe que tinha como responsabilidade efetuar testes de usabilidade.

O restante do capítulo é dividido como segue. A Seção 3.1 discute as necessidades discutidas durante o planejamento do projeto. A Seção 3.2 apresenta a visão geral do projeto, tratando de forma geral todo o fluxo de uma transação interligando os microsserviços com suas respectivas importâncias para desenvolvimento. Finalmente, a Seção 3.3 detalha cada microsserviço e suas funções.

3.1 Necessidades do projeto

No início do projeto, foram definidas diretrizes a serem seguidas a fim de uma boa experiência para o usuário durante o uso de tal sistema. Essas diretrizes foram:

- **Velocidade de resposta:** Um dos pilares do sistema era em relação a velocidade de resposta para o usuário sobre a respectiva transação, dando detalhes do status para o mesmo a partir do término do fluxo de pagamento, principalmente em transações do tipo Pix e cartão já que estas são conhecidas pela rápida compensação;

- **Monitoramento de falhas:** Em relação a problemas passados relatados por superiores, foi acordado que o estagiário precisava dar ênfase em questões de monitoramento para que qualquer falha fosse percebida, analisada, processada e resolvida;

- **Enfileiramento de requisição:** Era necessário evitar ao máximo sobrecargas por excesso de requisições advindas do *gateway* de pagamento (Safe2Pay), garantindo também que caso alguma dessas falhasse a mesma seria armazenada para reprocessamento;

- **Desenvolvimento seguro:** Uma plataforma segura contra ataques deveria ser priorizada, como por exemplo o uso de CDNs que já tratam questões de segurança para um desenvolvimento ágil e seguro com ênfase no que precisamos apenas;

- **Confiabilidade de dados:** Era de extrema importância a validação dos dados para que garantíssemos sempre que os dados em requisições por parte do *gateway* se mantivessem íntegros e confiáveis por consequência para que fossem realizadas as inscrições e o fluxo pudesse ocorrer normalmente até o fim por parte do sistema de inscrições;

3.2 Visão geral da Arquitetura

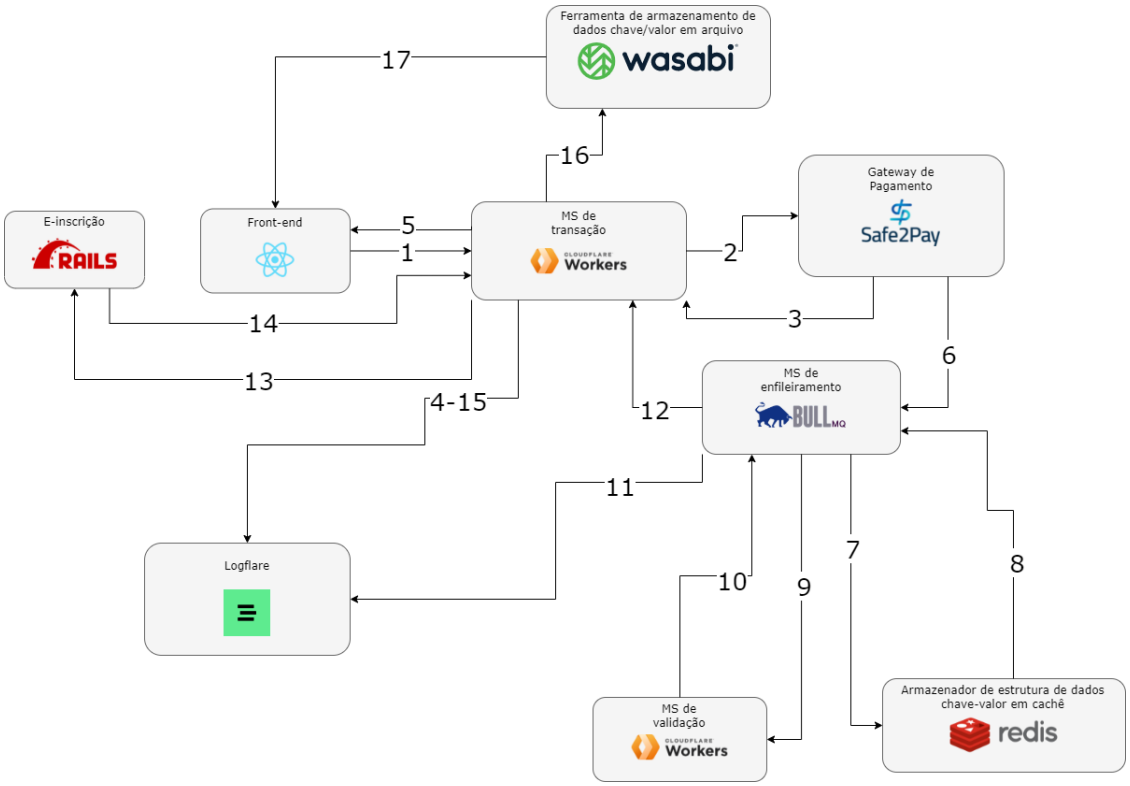
Se tratando de um sistema baseado na arquitetura de microsserviços, a equipe optou por desenvolver três microsserviços essenciais para o fluxo de pagamento: transação, enfileiramento e validação. O objetivo foi garantir maior segurança e controle para manutenção quando esta fosse requerida, reduzindo os riscos de falha. De acordo com os planejamentos foi utilizado a linguagem TypeScript que tem como base o JavaScript para desenvolver todos os microsserviços a fim de facilitar o desenvolvimento por conta da tipagem de dados e acesso a recursos mais recentes.

Sabendo que os microsserviços tem pontos de falha, planejou-se a adição de *logs* utilizando chamadas a ferramenta Logflare[1]. Esta ferramenta permite a visualização de *Logs* ao longo de tal fluxo. Sabendo disso, armazenaram-se informações e mensagens padronizadas para que a observação do fluxo de transações fosse contínua, permitindo a identificação de erros com precisão e um planejamento para correção.

[1] https://logflare.app

A Figura 3.1 apresenta a arquitetura proposta e o fluxo de transações contendo apenas uma enumeração do fluxo caso a transação ocorra por completo sem nenhuma falha. O fluxo se inicia com o recebimento dos dados através do *Front-end* (1). Encaminhando tais dados para o microsserviço responsável pelo controle principal da comunicação com o *gateway* de pagamento (Safe2Pay), sistema responsável pelo gerenciamento de inscrições e *Front-end*, e mapeando estes para envio ao *gateway* (2). Após isso ocorre o envio (3), onde caso a geração tenha sido um sucesso, tal microsserviço comunica o sistema de *Log* para armazenamento de informações importantes caso se tenha um problema futuramente (4). Logo após, é enviado como resposta ao *Front-end* uma *url* para que este possa monitorar via Wasabi, uma ferramenta de armazenamento e consulta rápida de dados chave/valor em arquivo (5). Pode-se ter como resposta um status "aguardando", garantindo que a transação foi gerada. Ou envia um status "negado"junto com o erro do sistema diretamente para o *Front-end* exibindo assim um parecer para o cliente sobre o motivo de tal falha. Este pode ser referente ao próprio microsserviço ou externamente, por algum tipo de bloqueio se tratando do *gateway* de pagamento.

Figura 3.1 – Figura exemplificando o fluxo da transação sem considerar erros



Fonte: Autor

Após a geração de tal transação, considerando que esta tenha ocorrido com sucesso, é necessária a atualização do sistema. Compensando a mesma no sistema responsável pela gestão de inscrições da empresa caso a transação seja autorizada, revelando assim que a mesma foi paga com sucesso. O microsserviço responsável por enfileirar a transação para processamento, utiliza da ação *postback* (6), filtrando as transações e enfileirando no Redis apenas as autorizadas para que não sobrecarregue o sistema (7).

Sabendo que tal *postback* foi enfileirado, no momento de seu processamento (8), o microsserviço responsável pelo enfileiramento dos *postbacks* faz uma requisição para o microsserviço responsável por validar os dados deste (9), recebendo assim sua permissão para prosseguir ou não com o processamento (10). Se o *postback* for aceito pelo microsserviço de validação, tal microsserviço solicita um novo *Log* para armazenamento de novas informações no Logflare (11). Enviando após isto

os dados de volta para o microsserviço responsável pelo controle principal das transações (12), onde receberá um tratamento de mapeamento para envio ao sistema de inscrições a fim de comunicar uma mudança de status do pagamento para autorizada para que se tome uma ação (13). Caso receba uma resposta positiva (14), o microsserviço comunica novamente com o Logflare informações pertinentes (15) e registra no Wasabi o sucesso (16) para que se possa comunicar ao cliente sobre seu acesso referente ao que foi pago através do *Front-end* (17). Caso tal resposta não seja positiva o microsserviço comunica o estorno automático para transações de cartão e pix para o *gateway* de pagamentos, registrando assim no Wasabi a mudança permitindo que os responsáveis pelo *Front-end* possam comunicar o erro para o cliente (16).

3.3 Microsserviço de transação

O primeiro microsserviço desenvolvido por parte do *Back-end*, foi hospedado no cloudflare Workers por oferecer a capacidade de desenvolver dando ênfase na lógica com todo suporte por parte de segurança desta.

Este é responsável por todo o fluxo principal do pagamento, sendo imbuído de gerar as transações no novo *gateway* de pagamento e tratar os erros comunicando sempre ao *Front-end* o sucesso da criação ou a falha caso este fosse motivado por um erro de criação da transação no *gateway*.

3.3.1 Geração de Transações

As transações são geradas a partir de dados cadastrais advindas do sistema e-inscrição e da captação de informações de pagamento servidas pelo próprio cliente em uma tela desenvolvida pelos responsáveis pelo *Front-end*, encarregado pela captação e envio de dados para o microsserviço. Tais transações eram geradas no *gateway* após um mapeamento para o formato aceito pelo mesmo dos dados recebidos pelo microsserviço.

O microsserviço envia como resposta um link responsável por armazenar e fornecer informações sobre o status da transação. Podendo ser um status positivo caso a mesma tenha sido aprovada e seguido até o fim do fluxo gerando assim uma inscrição para o cliente, ou negativa seguida de uma mensagem com o motivo desta caso tenha ocorrido alguma falha durante todo o processo.

3.3.2 Geração de inscrição

Com os dados já validados advindos do microsserviço responsável pelo enfileiramento, o microsserviço de transação realiza um mapeamento dos dados para o formato aceito pelo sistema de inscrições em eventos religiosos da empresa e os envia na forma de JSON para este, onde caso receba uma resposta positiva, confirmando que a inscrição foi gerada, o microsserviço registra no Wasabi a atualização confirmando para o *Front-end* que o pagamento foi aceito e o cliente pode prosseguir.

Se houver uma resposta negativa, advinda de algum erro em relação ao sistema de inscrição, o microsserviço comunica o estorno para o *gateway* de pagamento e informa ao *Front-end*, através do registro no Wasabi para esclarecimento do cliente, que o fluxo falhou e a transação foi estornada a fim de possibilitar uma nova tentativa de pagamento.

3.3.3 Armazenamento de status da transação

Durante o fluxo da transação, os microsserviços atualizam o estado da mesma na ferramenta Wasabi para que o *Front-end* possa consultar este status a fim de esclarecer como se encontra a transação, desde a sua criação, informando a pendência, até sua aceitação ou rejeição pelo sistema responsável pelas inscrições, após o pagamento.

A ferramenta foi escolhida por apresentar uma série de vantagens, sendo as principais pela sua velocidade de atualização e alta disponibilidade para consulta, permitindo assim um rápido retorno para o *Front-end* para que esse prosseguisse seu fluxo próprio.

3.4 Microsserviço de enfileiramento

Esse tipo de operação demanda um grande número de transações a serem processadas. Tais operações podem acarretar em uma sobrecarga no sistema responsável por gerar as inscrições, conforme esta fosse recebendo as confirmações pelo próprio microsserviço principal. Foi necessária a implementação de um segundo microsserviço responsável por enfileirar requisições, garantindo que estas não serão perdidas e serão processadas de acordo com o tempo imposto pelo desenvolvedor. Para isto usou-se a biblioteca BullMQ, gerenciadora de processamentos com regras de enfileiramento de requisições e o *Redis*, armazenador de estrutura de dados chave-valor em cachê. Devido a necessidade

de utilização do BullMQ e Redis, foi escolhido como hospedagem o *heroku* por oferecer suporte as ferramentas mencionadas.

Como o *postback* era enviado pelo *gateway* de pagamento a toda mudança de status da transação, este microsserviço ficou responsável por enfileirar apenas os *postbacks* de status "Autorizado" garantindo um fluxo sem resíduos de outros status. Após tais enfileiramentos o microsserviço processa o *postback* de acordo com o que foi especificado pelo estagiário.

Uma medida adotada para caso falhe o envio de *postback* por parte do *gateway* de pagamento, foi a implementação de *scripts* que rodavam diariamente para termos certeza que todas as compras pagas no *gateway* tiveram seus fluxos concluídos. Caso o *postback* não tenha sido enviado o *script* identifica isto e solicita um reenvio de *postback* para o *gateway* de pagamento.

3.5 Microsserviço de validação

Sabendo que a segurança deve ser um dos pilares do projeto, foi acordado que o estagiário precisava desenvolver um terceiro microsserviço. Para maior simplicidade, este foi hospedado no cloudflare workers.

O mesmo deveria ser um ambiente capaz de garantir a integridade dos dados essenciais para a transação, certificando assim que a mesma está de acordo com os dados apresentados no *gateway* de pagamento sem sofrer qualquer alteração em seu status de forma inesperada. Por fim, com isto, o ambiente seria capaz de eliminar a possibilidade de uma compensação errônea caso a transação tenha sido estornada por algum dos responsáveis.

3.6 Observação e manutenção

Após a entrega do microsserviço, é muito importante acompanhar o andamento do mesmo oferecendo manutenção e confiabilidade no que foi feito. Uma estratégia importante no acompanhamento de aplicações web, tanto *Back-end* quanto no *Front-end*, é o processo de análise de *Log*.

A análise de *Logs* permite a detecção de erros, monitorando o funcionamento dos microsserviços de forma personalizada de acordo com o que for configurado pelo desenvolvedor. Esta pode servir como respaldo de dados garantindo a integridade deste, já que estes ficam armazenados em

forma de *Log*. Uma das ferramentas utilizadas é a Logflare, que permite a configuração de um *Log* durante o desenvolvimento. Sendo utilizado para um monitoramento contínuo do que ocorre no sistema. Pode-se configurar este para armazenar tais *logs* em outras plataforma de armazenamento em nuvem de acordo com o que se tem de acesso pela empresa utilizadora.

Tais *Logs*, após apontarem erros, podem requisitar *scripts*. Estes são usados como forma de manutenção visando resolver tais anomalias do sistema. Aliado aos *scripts* de manutenção, podemos ter aqueles que rodam de forma automatizada diariamente visando verificar a integridade ao longo do fluxo, garantindo que caso os *Logs* falhem, estes serão capazes de detectar e comunicar aos desenvolvedores para que estes possam corrigir tais falhas.

3.7 Considerações Finais

O projeto foi desenvolvido em três meses, tendo seu uso validado durante dois meses. O mesmo se mantém em vigência contendo todo o fluxo principal de transações da empresa. Inicialmente alguns impasses foram encontrados se tratando de limitações não documentadas impostas pelo próprio *gateway* de pagamento, o que gerou uma série de tratamentos por parte da equipe e do estagiário, servindo como um grande aprendizado para este em relação a ferramenta utilizada.

Durante a graduação, o estagiário teve a oportunidade de participar da empresa júnior Emakers, onde obteve grande parte do conhecimento que permitiu um rápido desenvolvimento dentro da empresa E-inscrição. Sendo assim, esta foi crucial para agregar em tomadas de decisões durante o desenvolvimento no estágio.

A fim de manter uma organização no projeto seguindo os padrões da empresa, algumas diretrizes processuais foram submetidas na área de *tech*, sendo estas:

- O objetivo do projeto deveria ser avaliado e apresentado na reunião de OKR's sem possibilidade de remoção ou acréscimo até o final do trimestre;
- Toda semana era necessário a apresentação do que foi feito para o *tech Lead* a fim de avaliar se o projeto estava bem dimensionado ou não;

- Mesmo que sem a fidelidade se referida ao modelo original, foram pedidos uma boa ambientação de desenvolvimento garantindo que os testes seriam feitos em seus respectivos locais sem afetar o produto oficial após sua conclusão.

- Todos os testes e validações precisavam ser aprovados por uma equipe a parte antes da liberação de correções. Com isso em mente, uma equipe especializada em adquirir melhorias referentes ao cliente, captava todas as ideias e submetia através de reuniões para a equipe, a fim de que em novas *sprints*, período de tempo limitado para desenvolvimento de um produto, essas pudessem ser avaliadas e desenvolvidas.

Após a estabilização, o produto foi aperfeiçoado buscando otimização e segurança com autenticação mais otimizada. Algumas técnicas de validação de dados também foram adicionadas a fim de destacar os sistemas que se comunicaram com este, garantindo assim mais uma camada de integridade de dados e segurança também.

4 CONCLUSÃO

Este documento apresentou um relato das atividades desenvolvidas pelo estagiário na organização E-inscrição. O relatório dá ênfase nas atividades relacionadas ao desenvolvimento de um sistema baseado em arquitetura de microsserviços para o processamento de transações de pagamento. No entanto, o estagiário também realizou outras atividades referente ao desenvolvimento de diversos microsserviços. Estes foram desenvolvidos visando automatizar algumas tarefas feitas para monitoramento na empresa de serviços antigos e mais robustos.

O estagiário foi capaz de testar diversos serviços pedidos pela empresa apresentando relatórios contendo pontos positivos e negativos do mesmo, explorando assim novas tecnologias.

Este estágio agregou bastante no entendimento do estagiário acerca da regra de negócio da empresa como um todo, sendo esta muito complexa por conta da quantidade de comunicação necessária entre sistemas. Com todos os imprevistos encontrados, foi possível realizar a troca de conhecimentos complementando os conhecimentos prévios do estagiário em relação a desenvolvimento, facilitando assim na manutenção do sistema e na sua maturidade como desenvolvedor. Dos conhecimentos aprendidos no curso de Bacharelado em Ciência da Computação, alguns que contribuíram mais diretamente com as atividades desenvolvidas no estágio foram:

- Introdução a algoritmos, pela base em relação a lógica e o olhar crítico sobre o código evitando assim futuros problemas;
- Complexidade de algoritmos, pela capacidade de otimizações no geral através de uma sensibilidade maior no uso de algoritmos, garantindo que este escalasse bem por tal desempenho;
- Sistemas operacionais e estrutura de dados foram úteis por conta de estruturas de enfileiramento e tratamento de concorrência utilizado em armazenamentos simultâneos no mesmo arquivo, permitindo assim que o fluxo tivesse sempre uma ordem estável a ser seguida com maior segurança;

Apesar da qualidade de ensino em todas as matérias que o estagiário cursou durante a graduação, o curso poderia ter dado uma base melhor em relação a desenvolvimento Web, expandindo assim as possibilidades de conhecimento e mercado por parte de seus alunos. Tratando-se do estágio, no início o estagiário teve problemas em se adaptar devido a um ritmo intenso de *start up*, sendo assim

interessante uma melhor integração de novos estagiários. No entanto, até o término do estágio, foram percebidas melhorias promissoras.

REFERÊNCIAS

BARSOTI, N.; GIBERTONI, D. Impacto que o sequelize traz para o desenvolvimento de uma api construída em node. js com express. js. **Revista Interface Tecnológica**, v. 17, n. 2, p. 231–243, 2020.

BERTUOL, M. R. **Prado PHP Framework: um estudo experimental para controle de acesso a sites**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2011.

BULLMQ. **Groups**. 2022. Disponível em: <https://docs.bullmq.io/bullmq-pro/groups>. Acesso em: 02 de abril, 2022.

BULLMQ. **What is BullMQ**. 2022. Disponível em: <https://docs.bullmq.io/what-is-bullmq>. Acesso em: 19 de abril, 2022.

CLOUDFLARE, I. **Nossa história**. 2022. Disponível em: <https://www.cloudflare.com/pt-br/our-story/>. Acesso em: 18 de abril, 2022.

CLOUDFLARE, I. **O que é uma CDN?** 2022. Disponível em: <https://www.cloudflare.com/pt-br/learning/cdn/what-is-a-cdn/>. Acesso em: 18 de abril, 2022.

CUNHA, M. B. Entendendo o uso do git em equipes de desenvolvimento de software. 2018.

DOCS, M. W. **Introdução Express/Node.** 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction>. Acesso em: 20 de abril, 2022.

EMER, J. C. Migrando aplicações web para plataformas abertas: um estudo de caso. 2013.

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

FONSECA, R.; SIMÕES, A. Alternativas ao xml: Yaml e json. 2007.

GITHUB. **The 2021 State of the Octoverse**. 2021. Disponível em: <https://octoverse.github.com/>. Acesso em: 17 de março, 2022.

HEROKU. **The Heroku Platform**. 2022. Disponível em: <https://www.heroku.com/platform>. Acesso em: 16 de abril, 2022.

MALAV, B. **Microservices vs Monolithic architecture**. 2017. Disponível em: <https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4>. Acesso em: 28 de março, 2022.

MARQUES, A. I. A. **Desenvolvimento de API para aplicação cloud**. Tese (Doutorado), 2018.

MAZLAMI, G.; CITO, J.; LEITNER, P. Extraction of microservices from monolithic software architectures. In: **2017 IEEE International Conference on Web Services (ICWS)**. [S.l.: s.n.], 2017. p. 524–531.

MICROSOFT, I. **What is TypeScript?** 2022. Disponível em: <https://www.typescriptlang.org>. Acesso em: 19 de abril, 2022.

MONTEIRO EDUARDA, R. **et al. DevOps**. 1. ed. Disponível em: Minha Biblioteca: Grupo A, 2021.

MUSIL, S. **Record-breaking DDoS attack in Europe hits 400Gbps**. 2014. Disponível em: <https://www.cnet.com/news/privacy/record-breaking-ddos-attack-in-europe-hits-400gbps/>. Acesso em: 15 de abril, 2022.

OLIVEIRA, C.; ZANETTI, H. **Node.js: programe de forma rápida e prática**. Saraiva Educação S.A., 2021. ISBN 9786558110217. Disponível em: <https://books.google.com.br/books?id=QnRIEAAAQBAJ>.

OLIVEIRA, M. L. R. d. et al. Desenvolvimento do sistema web chemistery ênfase no back-end. Instituto Federal Goiano, 2019.

PAGAR.ME. **O que é o produto de Gateway?** 2022. Disponível em: <https://pagarme.zendesk.com/hc/pt-br/categories/200401679-Sobre-o-Pagar-me>. Acesso em: 20 de abril, 2022.

SANCHEZ-PALENCIA, G. **Gerenciando o fluxo assíncrono de operações em NodeJS**. 2019. Disponível em: <https://medium.com/@gustavospalencia/o-desenvolvimento-de-aplicaç~oes-nodejs-aumenta-a-cada-dia-talvez-pela-facilidade-de-dc9eb47ab8b5>. Acesso em: 03 de abril, 2022.

SOARES, M. dos S. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, v. 3, n. 1, 2004.

VILLAÇA, L.; JR, A. F. P.; AZEVEDO, L. G. Construindo aplicações distribuídas com microsserviços. **Tópicos em Sistemas de Informação: Minicursos XV Simpósio Brasileiro de Sistemas de Informação. SBC**, 2018.