UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Departamento de Sistemas de Computação

Plataforma pedagógica para aprendizado baseado em projeto na área de computação distribuída e baseada em serviços.

Felipe Tiago De Carli

São Carlos - SP

Plataforma pedagógica para aprendizado baseado em projeto na área de computação distribuída e baseada em serviços.

Felipe Tiago De Carli

Orientador: Francisco Jose Monaco

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina Sistemas Distribuídos e Programação Concorrente do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP para obtenção do título de Engenheiro de Computação.

Área de Concentração: Computação distribuída baseada em serviços

USP – São Carlos
10/11/2021

“O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.”.

(José de Alencar – escritor)

Agradecimentos

Agradeço aos meus pais, Laércio De Carli e Edini De Carli, que me forneceram a oportunidade de atingir qualquer objetivo que eu pudesse imaginar.

Agradeço a minha irmã, Caroline De Carli, que demonstrou seu apoio em momento difíceis, dividindo muitas vezes seu espaço de estudo, me ajudando em tantas maneiras que é impossível enumerá-las.

À Luah Debus, minha parceira, amiga e namorada, que algumas vezes longe, sempre esteve ao meu lado, nos momentos bons e ruins, em todos os anos da minha graduação.

Ao Robson Barbosa, Daniel Moser e Daniela Horta, meus amigos e colegas de trabalho, que, além de demonstrarem paciência e permitirem a conciliação da minha graduação com o trabalho, me ofereceram uma grande oportunidade para meu crescimento pessoal.

Aos meus amigos Rafael Bariccatti e Gabriel Dezan, que estiveram comigo durante a graduação, me encorajando a seguir em frente em qualquer obstáculo encontrado.

Agradeço também ao meu orientador, Francisco Jose Monaco, por ter me dado a oportunidade de estudar um tema atual, alinhado aos meus objetivos pessoais e profissionais.

Resumo

A crescente necessidade de desenvolver softwares complexos de maneiras mais rápidas e seguras trouxe novos métodos que apoiam uma construção mais sustentável de códigos. Dentre esses métodos, pode-se citar a metodologia de Microsserviços, que possui como base alguns dos fundamentos da Computação Distribuída. Mesmo com o sucesso dos Microsserviços, ainda há muitas empresas e desenvolvedores que não a utilizam. Por isto, este trabalho implementa um sistema educacional aberto para o aprendizado e desenvolvimento prático de arquiteturas baseadas em Microsserviços. Para a plataforma desenvolvida, foram propostos métodos avaliativos para determinar a grau de conhecimento do usuário do sistema, assim como fornecer um grau de visibilidade para que se possa observar onde se encontra suas dificuldades.

Índice

CAPÍTULO 1: INTRODUÇÃO

1.1. Contextualização e Motivação

A partir da década de 70, houve um aumento expressivo no número de novas pesquisas na área de design de softwares e das consequências da escolha de certos métodos de desenvolvimento [2]. A partir dessa época, diversas publicações a respeito do tópico trouxeram definições formais de técnicas e ferramentas que poderiam ser utilizadas para o desenvolvimento mais saudável de produtos na indústria do desenvolvimento de software.

No início do milênio, houve o surgimento de métodos de engenharia de software baseado em componentes, que mais tarde se desenvolveram para SOAs (Service-oriented architectures) [7]. Essas novas arquiteturas trazem o conceito de serviço como uma unidade desacoplada de um programa maior, que oferece funcionalidades a outros componentes através de troca de mensagens [2]. Considerada por Dragoni (2017, p. 201) como a "primeira geração" de arquiteturas orientadas a serviços, essa metodologia trouxe vários benefícios, como o dinamismo, modularidade e reusabilidade, desenvolvimento distribuído e integração com sistemas legado.

Mesmo com seus benefícios, a primeira geração de SOAs definiu requerimentos bastante complexos e desafiadores para suas implementações [3], o que impediu a adoção generalizada desse paradigma pela indústria de software.

Devido a fraca adoção da primeira geração de SOAs, uma nova proposta de arquitetura orientada a serviços se tornou popular no meio acadêmico. Essa nova proposta, denominada por Erl como Segunda Geração de SOAs, trouxe consigo a ideia de componentização – conceito já existente na tecnologia de Programação Orientada a Objetos.

A Segunda Geração de SOAs uniu os benefícios já existentes da primeira geração, com o conceito de *deployment* independente e *business capability* [9]. Com a junção desses

conceitos e a remoção das complexidades desnecessárias da primeira geração de SOAs, formou-se o que se conhece hoje como arquitetura de Microsserviços.

Os benefícios da arquitetura de Microsserviços são já bastante difundidos no mercado. Um estudo de D. Taibi et al. 2017, realizou uma pesquisa com diversos profissionais com 10 ou mais anos de experiência no mercado de TI, e extraiu as três vantagens mais significativas para os profissionais consultados:

- Microsserviços possuem uma escalabilidade muito maior do que arquiteturas monolíticas tradicionais.
- Times de desenvolvimento podem focar em somente um microsserviço, sem necessidade de reter responsabilidade por outra área do negócio.
- Microsserviços podem ser lançados de forma independente entre si, o que acelera o processo de desenvolvimento e permite um retorno mais rápido do cliente.

Mesmo com os benefícios da arquitetura de microsserviços serem bastante claros, várias empresas ainda hesitam em migrar para esse novo paradigma de desenvolvimento, devido ao fato de que acreditam ser apenas uma "onda momentânea", ou porque não estão confiantes em realizar o processo de migração e não estão totalmente convencidos dos benefícios [10].

Devido à essa insegurança e a falta de confiança e conhecimentos das companhias e de alguns desenvolvedores, este trabalho tem por objetivo oferecer uma metodologia de ensino prático de arquitetura de Microsserviços, de modo que se possa avaliar o conhecimento do praticante conforme o cumprimento dos requisitos das atividades apresentadas.

1.2. Objetivos

Este trabalho tem como objetivo propor uma plataforma pedagógica em código aberto, onde alunos interessados no aprendizado de microsserviços poderão acessar e

interagir. Será proposto um ambiente simulado, baseado na arquitetura de microsserviços, onde o aluno deverá avaliar e corrigir possíveis problemas, de forma com que o ambiente seja o mais coerente possível com a arquitetura proposta.

O intuito geral deste trabalho é fornecer um apoio ao aprendizado de microsserviços, de forma com que a sua utilização seja mais difundida, e também ofereça uma maior confiança e segurança para desenvolvedores em empresas que desejam implantar esta nova arquitetura.

1.3. Organização do Trabalho

O presente trabalho está organizado em 4 diferentes capítulos. O primeiro capítulo é o capítulo presente (Introdução), onde é abordado o contexto em que o tema de microsserviços está presente. Neste capítulo, também é oferecido uma visão macro do trabalho ao leitor.

No CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA, é disposto as principais características e metodologias da arquitetura de microsserviços disponíveis hoje na literatura. Também, é abordado temas sobre recursos educacionais abertos, já que possui uma forte presença no objetivo deste trabalho.

No CAPÍTULO 3: DESENVOLVIMENTO DO TRABALHO, é apresentado os métodos utilizados para o desenvolvimento do ambiente proposto, as metodologias para a interação com o ambiente, e os mecanismos para que a avaliação do aluno participante possa ser realizada.

No CAPÍTULO 4: CONCLUSÃO, é disponibilizado uma conclusão a respeito da eficácia no uso e aprendizado com o ambiente proposto, obtidas a partir de testes com usuários reais e simulados.

CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

2.1. Considerações Iniciais

Neste capítulo apresenta-se os principais conceitos e terminologias utilizadas nos trabalhos existentes na área de microsserviços e recursos educacionais abertos. Dentre os tópicos dispostos, é abordado o tópico de implantação, migração e arquitetura de microsserviços. Ademais, cita-se tópicos de microsserviços em ambientes Cloud, por se tratar de uma prática bastante presente na arquitetura de microsserviços.

2.2. Microsserviços e Padrões de Arquitetura

Segundo Taibi (2017), microsserviços são pequenas aplicações que podem ser implantadas, escaladas e testadas de maneira independente, além de possuírem uma única responsabilidade. Shadija e Rezai complementam essa definição, propondo que cada microsserviço deve endereçar uma funcionalidade de negocio única, de maneira com que o conjunto dos microsserviços possam ser capazes de exercer uma função de negócios maior.

De forma geral, uma das únicas características que os microsserviços devem compartilhar entre si é a definição de um protocolo de comunicação, sendo geralmente utilizado o protocolo Representational State Transfer (REST) ou Message Queues (MQ) [9]. Esse único requerimento de definição de um protocolo de troca de informações e necessário, visto que uma das definições de microsserviços é que cada um possui sua própria fonte de dados – qualquer necessidade de obter dados de outro microsserviço, deve ser realizado uma comunicação através do protocolo pré-definido. A figura 1 demonstra um diagrama exemplo de uma arquitetura de microserviços.

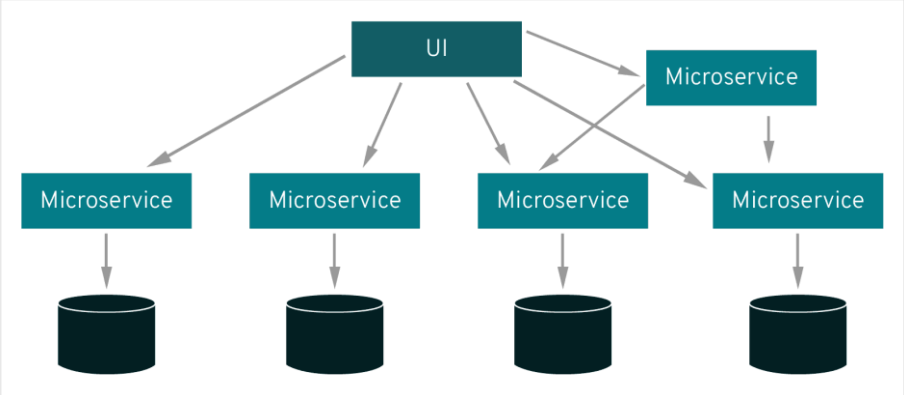

Figura 1 – Diagrama ilustrando uma arquitetura de Microsserviços Fonte: https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh

É necessário, portanto, definir e planejar o escopo de cada microsserviço. A correta divisão de funções é o desafio principal da arquitetura de microsserviços, desafio este que tem por objetivo trazer todos os benefícios de escalabilidade e independência que o novo paradigma de microsserviços propõe.

Diversos padrões de microsserviços surgiram ao longo dos anos. Os padrões propostos tem como objetivo auxiliar desenvolvedores e arquitetos de softwares a encontrar soluções adequadas a seus modelos de negocio. Os padrões aqui descritos foram extraídos do artigo Architectural Patterns for Microservices: A Systematic Mapping Study, de D. Taibi, V. Lenardurzzi e C. Pahl, publicado em 2018.

Os três diferentes padrões de arquiteturas descritos por Taibi são divididos em três categorias:

- Orientado a Orquestração e Orientado a Coordenação: Esse padrão de arquitetura, segundo Taibi, tem como principal característica a captura de comunicação e coordenação segundo uma perspectiva logica.

- Baseado em estratégias de implantações: Padrão de arquitetura voltado a implantação de microsserviços que atuam dentro de containers ou maquinas virtuais.

- Baseado em gerenciamento de dados: Padrões de arquitetura que se baseiam nos diferentes desafios e opções no que se refere ao armazenamento de dados.

Nas subseções apresentadas a seguir, serão discutidos e exemplificados os padrões de arquitetura Orientado a Orquestração e Orientado a Coordenação.

2.2.1. Padrão API-Gateway

Neste padrão de arquitetura, os microsserviços realizam suas funções para outros serviços através de uma API. Esse padrão é utilizado para fornecer um ponto focal para tratar requisições de usuários finais de uma aplicação, definido por Taibi como um mecanismo de agregação ao lado do servidor.

O API Gateway, comentado anteriormente, é o ponto inicial de comunicação do usuário final com os microsserviços. Qualquer comunicação que deve ser estabelecida com um microsserviço deve ser antes tratada e roteada pelo API Gateway. A função desse mecanismo e também tratar autenticação, transformação de protocolos e definir limites de requisições.

De maneira geral, e possível atribuir a responsabilidade de balanceamento de carga para o API Gateway, visto que este conhece a localização e endereço de todos os microsserviços da aplicação.

Na figura 2 é possível visualizar um diagrama que representa um exemplo da arquitetura API Gateway. Nesse exemplo, é considerado uma aplicação simples de e-commerce, onde há os seguintes microsserviços:

- Microsserviço de Usuário: Responsável por processar e armazenar informações a respeito de usuários da aplicação.

- Microsserviço de Compras: Responsável por processar e armazenar informações a respeito de compras realizadas no e-commerce.

- Microsserviço de Catálogo: Responsável por armazenar e distribuir informações a respeito dos diferentes itens a venda no e-commerce.

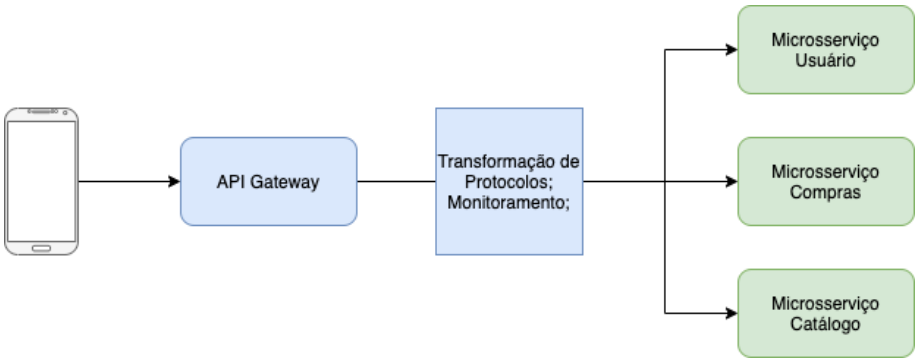

Figura 2 - Diagrama exemplo da arquitetura API Gateway. Fonte: Autor

No exemplo ilustrado pela figura 2, os itens em azul são referentes ao processamento e comunicação realizada pelo API Gateway. Os itens em verde são as representações dos microsserviços de exemplo apresentados anteriormente.

2.2.2. Padrão Service Discovery

O padrão Service Discovery tem por objetivo solucionar problemas comuns na implantação de microsserviços em ambientes Cloud [8].

Em uma implantação em máquinas físicas (instâncias bare metal), os endereços de rede são geralmente estáticos e bem definidos. Para realizar a comunicação ente os microsserviços, os programas podem fazer uma leitura em um arquivo que contém os endereços de cada máquina.

No caso de implantações mais modernas, como em containers em um ambiente Cloud, os endereços passam a ser dinâmicos e podem mudar constantemente, o que torna difícil manter uma lista de endereços em um arquivo.

No caso do padrão Client-Side Discovery, o cliente, antes de realizar requisições aos microsserviços, é responsável por consultar um serviço de registros, em que é armazenado as informações de rede dos microsserviços disponíveis. Após consultar o registro, o cliente utiliza um balanceamento de carga próprio e realiza a requisição ao microsserviço desejado.

De acordo com Richardson, a principal vantagem dessa arquitetura é a facilidade de desenvolvimento, já que a complexidade do roteamento para os microsserviços é de responsabilidade do cliente.

Um diagrama do padrão de arquitetura Client-Side Discovery pode ser visualizado na figura 3.

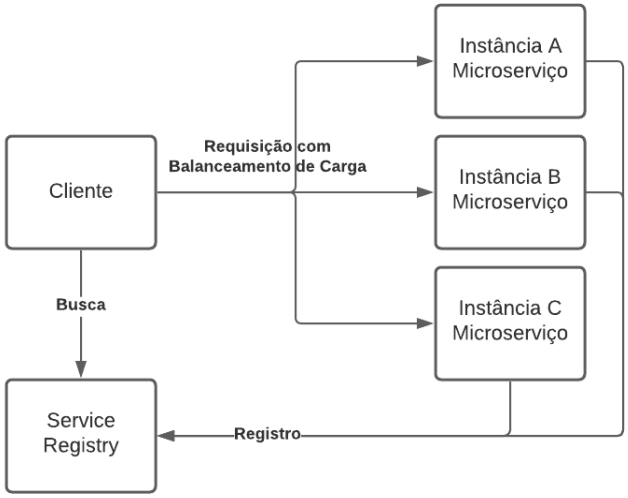

Figura 3 - Arquitetura Client-Side Discovery. Adaptado de https://microservices.io/patterns/client-side-discovery.html

Para o padrão Server-Side Discovery, o cliente não necessita fazer a consulta em um registro, e realiza a requisição diretamente a um balanceador de carga. O balanceador de cargas, ao receber uma requisição, realiza uma consulta no serviço de registros, onde extrai os endereços disponíveis de microsserviços. Após a consulta e a seleção de um endereço, o balanceador de cargas encaminha a requisição para o microsserviço desejado.

Um diagrama do padrão de arquitetura Server-Side Discovery pode ser visualizado na figura 4.

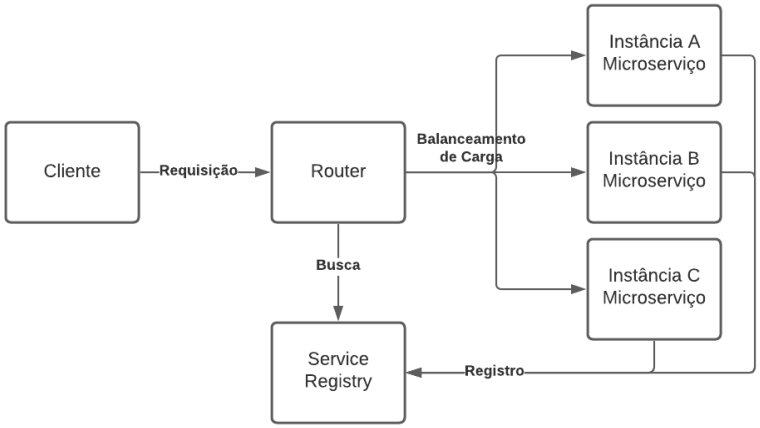

Figura 3 - Arquitetura Server-Side Discovery. Adaptado de https://microservices.io/patterns/server-side-discovery.html

2.2.3. Padrões Híbridos

O padrão de arquitetura híbrido, como definido por Taibi, atua de forma semelhante aos padrões API Gateway e Service Discovery, porém há a substituição de um API Gateway por um Message Bus.

O uso do Message Bus, ao invés do API Gateway, fornece uma camada com possibilidade de comunicação assíncrona, o que se torna extremamente útil em casos onde há um grande fluxo de conexões instáveis, como no caso de dispositivos IOT em locais remotos [12].

2.3. Microsserviços e Migrações

Com o crescimento e adoção dos microsserviços, diversas empresas têm optado por realizar uma migração de seus sistemas monolíticos, realizando uma refatoração de seus sistemas back-end para atender os novos paradigmas da arquitetura [6],

Durante o processo de migração, é necessário considerar quais componentes do sistema monolítico possuem a maior frequência de uso. Componentes que são frequentemente utilizados devem ser priorizados na migração para a arquitetura de microsserviços [4].

Por conta das diferentes possibilidades que um sistema monolítico pode estar implementado, a migração de um sistema monolítico para microsserviços não é um processo trivial, e, portanto, não há como definir um conjunto de passos que pode ser aplicado para qualquer situação [1].

Balalaie apresenta uma série de padrões de migrações que podem ser utilizados dependendo do sistema monolítico a ser migrado. Para cada padrão, é associado uma intenção e uma situação de reuso, assim como o problema atual da arquitetura monolítica. As associações, segundo Balalaie, devem ser utilizadas (em conjunto ou não) para que se possa escolher o processo de migração de maneira efetiva.

Com o grande número de migrações de aplicações monolíticas para arquiteturas de microsserviços, diversas empresas optaram por publicar seus maiores desafios e complicações durante o processo de migração.

Taibi, a partir de um conjunto de publicações de jornadas de migrações para microsserviços, propôs um compilado com os problemas mais frequentes encontrados pelas equipes durante o processo de migração, descritos a seguir:

- Migração do Banco de Dados: Foi reportado que a migração de banco de dados legados para uma arquitetura de microsserviços trouxe bastante complexidade ao processo de migração como um todo. Muitas vezes, por conta do tamanho e complexidade dos dados, bancos de dados foram mantidos em sua arquitetura original, mantendo um sistema de microsserviços com banco de dados compartilhados.

- Comunicação entre microsserviços: A escolha de protocolos de comunicação entre microsserviços trouxe uma complexidade a mais no processo de migração, devido ao fato de que há uma adição na complexidade do desenvolvimento de código para que se possa atender esse requisito.

- Estimativa de Esforços: Foi verificado que a previsão de esforços para o desenvolvimento de uma arquitetura de microsserviços diverge bastante da arquitetura monolítica. Taibi apresenta um aumento de 20% de tempo de

desenvolvimento na implementação de uma arquitetura de microsserviços em comparação com uma monolítica.

- Conversão de Livrarias: Taibi identifica esforços complexos para o reuso de livrarias que estavam sendo utilizadas na arquitetura monolítica. Em alguns casos, há a necessidade de uma refatoração e conversão da livraria para que se possa ser utilizada na arquitetura de microsserviços.

- Infraestrutura e DevOps: É identificado um esforço considerável na implementação e implantação de uma infraestrutura adequada para o uso de microsserviços. Nesses casos, Taibi identifica a necessidade do uso de DevOps para realizar automatizações e implantações automáticas.

2.4. Trabalhos Relacionados

2.4.1. Analisando a história dos microsserviços: Dragoni e sua análise histórica e de desafios futuros dessa área.

Através de uma revisão bibliográfica, Dragoni (2017) realiza uma releitura sobre os microsserviços, desde sua história, com o estudo que vem de algumas décadas da engenharia de software. O autor também descreve as principais característica da arquitetura de microserviços, impondo suas principais vantagens e desvantagens no sucesso do desenvolvimento de um sistema.

2.4.2 Analisando a compreensão dos microserviços: Shadija e o rumo a compreensão dos microserviços.

Em seu estudo, Shadija (2017) analisa as abordagens atuais para o desenvolvimento de uma arquitetura de software baseada em microserviços. Comparando a modalidade de Arquitetura orientada ao serviço (SOA) com os microsserviços, Shadija descreve características próprias de cada abordagem.

O autor termina sua obra descrevendo a importância e a possibilidade do uso de microsserviços no desenvolvimento de aplicações escaláveis e flexíveis a novas funções, conforme ocorre diferentes variações de negócios.

2.4.3 Uma investigação sobre os processos, motivações e dificuldades na migração para a arquitetura de microsserviços por Taibi

Em seu estudo, Taibi (2017) realiza uma descrição dos microserviços como um todo, focando em sua metodologia de implantação e comunicação.

Durante o trabalho o autor fornece uma comparação dos processos de migração de arquitetura monolíticas para microserviços, realizando uma classificação de motivações e problemas.

2.5. Considerações Finais

É possível notar que a leitura sobre os autores citados trouxe uma nova visão a respeito dos processos de imigração e da função dos microsserviços. Após esse capítulo, observaremos a descrição do processo de desenvolvimento desse projeto visualizando detalhadamente a tecnologia e os objetivos de aprendizado envolvidos.

CAPÍTULO 3: DESENVOLVIMENTO DO TRABALHO

3.1. Considerações Iniciais

Neste capítulo descreveu-se o processo de desenvolvimento do projeto, de forma com que seja possível visualizar detalhadamente os aspectos de tecnologia e objetivos de aprendizado envolvidos, além do detalhamento das características dos códigos utilizados para o desenvolvimento da plataforma, assim como os método para realizar uma análise da interação do usuário com o sistema. Por fim, apresenta-se os objetivos e resultados esperados dos usuários com o uso da plataforma.

3.2. Descrição do Problema e Projeto

3.2.1. Problemas e Objetivos

A migração de sistema monolítico para uma arquitetura de microserviços é um processo cada vez mais comum na área de Engenharia de Software, e é sempre acompanhado de desafios e escolhas difíceis. Por isso, neste trabalho, foi desenvolvido um sistema monolítico base, com o objetivo de proporcionar a alunos uma situação simulada de um ambiente onde é necessário realização de uma migração para a arquitetura de microserviços.

Para atingir esse objetivo, foi também desenvolvido uma série de testes automatizados para verificar o funcionamento adequado de todas as operações do sistema monolítico desenvolvido. Espera-se, portanto, que o sistema em uma arquitetura de microserviços obtenha sucesso em absolutamente todos os testes.

Para realizar a migração do sistema, o aluno deve considerar que a aplicação é utilizada por uma aplicação Front-End legado, onde não há mais suporte de desenvolvimento. Portanto, as rotas definidas no sistema monolítico devem continuar retornando os resultados da mesma maneira, de forma com que não seja necessária uma restruturação do sistema Front-End legado.

3.2.2. Descrição do Sistema

O sistema monolítico desenvolvido representa uma aplicação de gerenciamento de produtos de varejo. O uso do sistema permite que seja possível realizar operações com usuários, produtos, pedidos, pagamentos e cupons de desconto.

O sistema foi arquitetado de maneira com que seja minimamente complexo, de maneira com que a migração para a arquitetura de microserviços já traga benefícios claros de leitura e manutenção da aplicação. Uma visão geral dos componentes e da arquitetura da aplicação pode ser visualizada na figura 5.

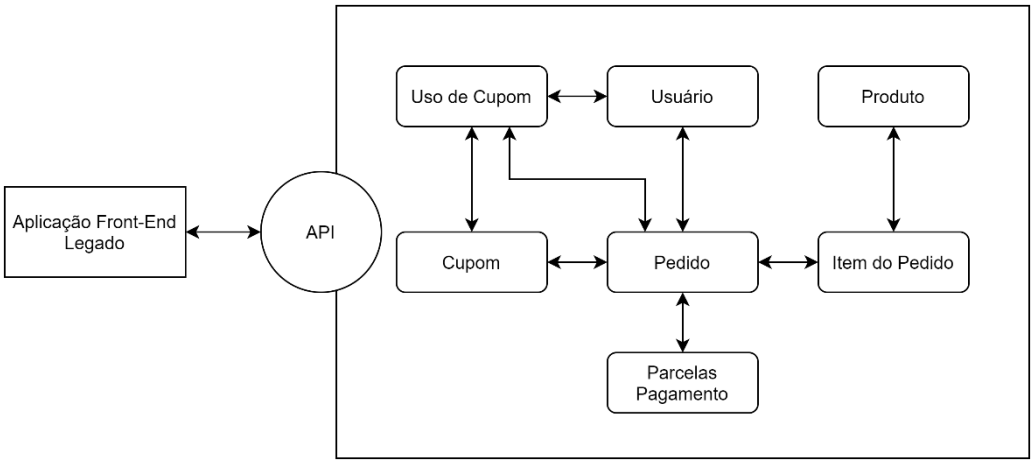

Figura 4 – Visão geral dos componentes da aplicação monolítica desenvolvida. Fonte: Autor.

O desenvolvimento da aplicação foi realizado utilizando a linguagem JavaScript, com o uso do *runtime* Node.js e do framework *express*.

A escolha do *stack* de tecnologia (JavaScript, Node.js e express), é devido ao fato da facilidade de leitura da aplicação, além de que a linguagem JavaScript é hoje a terceira linguagem mais utilizada no mundo [13]. Essas características fazem com que o aluno tenha uma maior facilidade em se adequar e entender o funcionamento do sistema, o que garante uma melhor experiência ao realizar o processo de migração para a arquitetura de microserviços.

3.2.3. Base de Dados

A tecnologia de banco de dados escolhida foi o PostgreSQL, uma engine de código livre mantida pela comunidade.

Foi realizado uma modelagem considerando a arquitetura monolítica da aplicação, o que implica na não distribuição das tabelas em mais bases de dados diferentes. O diagrama Entidade-Relacionamento desenvolvido pode ser visualizado na figura 6.

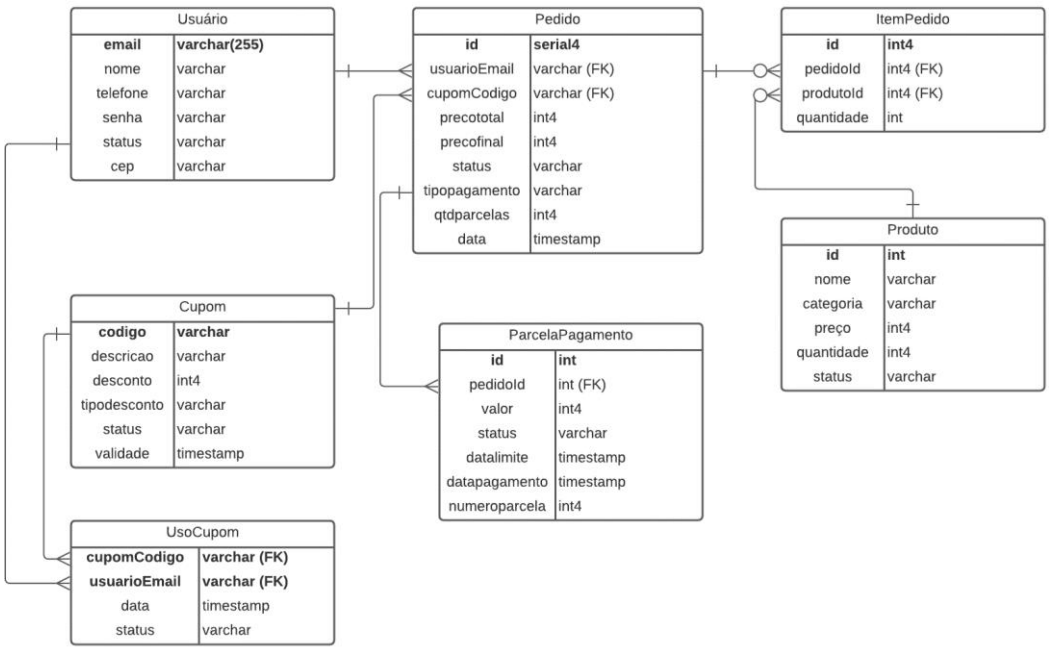

Figura 5 - Diagrama Entidade-Relacionamento da aplicação desenvolvida. Fonte: Autor.

3.2.4. Módulos da Aplicação

A aplicação desenvolvida é composta por 7 módulos diferentes, além do API Gateway para realizar a comunicação com chamadas externas. Cada módulo possui sua tabela correspondente no banco de dados, e interage com outros módulos em situações específicas. Os itens a seguir descrevem cada módulo e suas respectivas funcionalidades e interações com o sistema.

3.2.4.1. Módulo Usuário

O módulo USUÁRIO possui funcionalidades de recuperação, criação, atualização e remoção.

- Criação (POST /api/users): Realiza a criação de um novo usuário no banco de dados. Os parâmetros nome, telefone, senha e email são necessários para a chamada.

- Recuperação (GET /api/users): Recupera uma lista de usuários ou um usuário específico no banco de dados.

- Atualização (PUT /api/users): Altera informações do usuário e salva no banco de dados.

- Remoção (DELETE /api/users): Marca o usuário como “inativo”.

3.2.4.2. Módulo Produto

O módulo PRODUTO possui funcionalidades de recuperação, criação, atualização e remoção.

- Criação (POST /api/products): Realiza a criação de um novo produto no banco de dados. Os parâmetros nome, categoria, preço e quantidade são necessários para a chamada.

- Recuperação (GET /api/products): Recupera uma lista de produtos ou um produto específico do banco de dados.

- Atualização (PUT /api/products): Altera informações do produto e salva no banco de dados. Caso o parâmetro *retirar* seja enviado com valor *“true”*, a rota aceita apenas a opção *quantidade*, e a aplicação atualizará os valores de quantidade realizando a subtração da quantidade atual de produtos e a quantidade fornecida nos parâmetros. Se a quantidade informada for maior

do que a quantidade disponível, é retornado um erro. Se o produto atingir a quantidade 0, o status do produto é marcado como “indisponível”.

- Remoção (DELETE /api/products): Marca o produto como “inativo”.

3.2.4.3. Módulo Cupom

O módulo CUPOM possui funcionalidades de recuperação, criação e atualização.

- Criação (POST /api/coupons): Realiza a criação de um novo cupom no banco de dados. Os parâmetros *codigo*, *desconto* e *tipodesconto* são necessários para a chamada.

- Recuperação (GET /api/coupons): Recupera uma lista de cupons ou um cupom específico do banco de dados.

- Atualização (PUT /api/coupons): Altera informações do cupom e salva no banco de dados.

- Remoção (DELETE /api/coupons): Marca o cupom como “inativo”.

3.2.4.4. Módulo Uso-Cupom

O módulo CUPOM possui funcionalidades de recuperação, criação e remoção. Esse módulo é utilizado durante a criação e atualização de um pedido, quando um código de cupom é informado. O módulo CUPOM é responsável por guardar informações de uso de cupons, e prevenir a utilização de um mesmo cupom por um usuário mais de uma vez.

- Criação (POST /api/couponUsages): Cria uma nova utilização de cupom e armazena no banco de dados. É necessário que seja fornecido o id do pedido e o e-mail do usuário que utilizou o cupom.

- Recuperação (GET /api/couponUsages): Recupera uma lista de uso de cupons ou um uso de cupom específico do banco de dados.

- Remoção (DELETE /api/couponUsages): Remove o uso de cupom do banco de dados, o que permite que o usuário possa utilizar o cupom novamente.

3.2.4.5. Módulo Pedido

O módulo PEDIDO possui funcionalidades de recuperação, criação e atualização.

- Criação (POST /api/orders): Cria um novo pedido e armazena no banco de dados. É necessário que seja informado o e-mail do usuário que criou o pedido. É possível também informar um código de cupom, que será utilizado para descontar o valor no preço final do pedido. Nesse caso, será criado uma entrada na tabela UsoCupom, através do módulo Uso-Cupom descrito anteriormente.

- Recuperação (GET /api/orders): Recupera uma lista de pedidos ou um pedido específico do banco de dados.

- Atualização (PUT /api/orders): Atualiza o pedido com novas informações. Caso seja informado o parâmetro "concluirPedido" como *true*, o pedido é finalizado e é calculado os valores das parcelas de pagamento, criando entradas de parcelas utilizando o módulo PARCELA-PAGAMENTO. O módulo ITEM-PEDIDO utiliza essa rota informando o parâmetro "precoAdicional", o que realiza um novo cálculo no valor total e final do pedido.

3.2.4.6. Módulo Item-Pedido

O módulo ITEM-PEDIDO possui funcionalidades de recuperação, criação e remoção.

- Criação (POST /api/orderitems): Cria um novo item para um determinado pedido e armazena no banco de dados. É necessário que seja informado o id do pedido, id do produto e a quantidade do produto a ser adicionado. Além da criação de uma entrada na tabela itempedido, é realizado uma chamada de atualização no módulo PEDIDO, como descrito anteriormente. Também é

realizado uma chamada de atualização no módulo PRODUTO, reduzindo o número de quantidade disponíveis do produto com base no valor *quantidade* informado.

- Recuperação (GET /api/orderitems): Recupera uma lista de item de pedidos ou um item de pedido específico do banco de dados.

- Remoção (DELETE /api/orderitems): Remove o item de pedido. Os valores de preço total e preço final são recalculados através de uma nova chamada ao modulo PEDIDO. Também é recalculado a quantidade de produtos disponíveis através de uma chamada de atualização no módulo PRODUTO.

3.2.4.7. Módulo Parcela-Pagamento

O módulo PARCELA-PAGAMENTO possui funcionalidades de recuperação, criação e atualização.

- Criação (POST /api/installments): Cria uma nova parcela de pagamento para um determinado pedido e armazena no banco de dados. É necessário que seja informado o id do pedido e o valor da parcela. A criação da parcela é chamada através do módulo PEDIDO, no processo de atualização, como descrito anteriormente.

- Recuperação (GET /api/installments): Recupera uma lista de parcelas, parcelas de um pedido específico, ou uma parcela específica do banco de dados.

- Atualização (PUT /api/installments): Atualiza a parcela com o status “aprovado” ou “rejeitado”. Caso seja a aprovação da última parcela de um pedido, é realizado uma chamada de atualização no módulo PEDIDO, alterando o status do pedido para “pago”.

3.2.5. Testes

Os testes automatizados são a parte crucial do desenvolvimento da aplicação. É através dos testes que é verificado o funcionamento correto de todos os módulos descritos na subseção anterior.

Para o desenvolvimento dos testes, foram realizadas chamadas diretas à API da aplicação. Dessa maneira, é possível utilizar os mesmos testes para confirmar o bom funcionamento do sistema em uma arquitetura de microserviços.

Os testes foram desenvolvidos utilizando a framework Jest, que fornece uma série de funcionalidades que permitem verificar se o resultado das operações foi satisfatório.

3.3. Descrição das Atividades Realizadas

3.3.1. Introdução

As atividades realizadas durante o desenvolvimento do trabalho foram focadas no planejamento e no desenvolvimento da aplicação descrita. A aplicação, banco de dados e os testes automatizados foram arquitetados de maneira com que seja possível e vantajoso uma migração para a arquitetura de microserviços.

Primeiramente, foi realizado um estudo da arquitetura monolítica, de forma com que a aplicação desenvolvida siga os padrões propostos por essa arquitetura.

Após o estudo dos padrões da arquitetura monolítica, foi iniciado o planejamento e arquitetura do banco de dados da aplicação. A arquitetura do banco de dados foi realizada utilizando diagramas relacionais e o diagrama entidade-relacionamento.

Em seguida, foi definido a tecnologia para o banco de dados e foi iniciado o processo de criação de uma instância em um container Docker, de maneira com que os dados sejam salvos mesmo após a parada do processo.

Após a conclusão da implantação do banco de dados, foi iniciado o desenvolvimento da aplicação, com base nos padrões de arquitetura monolítica pesquisadas no primeiro passo.

Os testes automatizados foram desenvolvidos após a conclusão do desenvolvimento da aplicação. Foi desenvolvido testes unitários e de integração, de maneira com que possam ser utilizados no ambiente desenvolvido e em um ambiente de microserviços.

Por fim, foram estabelecidas regras de avaliação para uma arquitetura de microserviços. Nessa fase, foi elaborado uma série de itens que deverão ser seguidos para que a aplicação tenha uma boa arquitetura de microserviços. Foi definido pesos para cada item elaborado, de forma com que se possa utilizar os itens para avaliar de forma quantitativa a qualidade da migração da aplicação para microserviços.

As subseções a seguir detalham o processo de desenvolvimento da segunda à última atividade realizada.

3.3.2. Arquitetura do Banco de Dados

Para o desenvolvimento da arquitetura do Banco de Dados, foi decidido a utilização do banco de dados relacional. O banco de dados relacional apresenta uma robustez muito grande e é o paradigma mais utilizado no mundo para implantação de bancos de dados.

Inicialmente, mapeou-se as entidades da aplicação, as quais são: Usuário, Pedido, Parcela de Pagamento, Item do Pedido, Cupom, Uso de Cupom e Produto. A partir dessas entidades e considerando seus relacionamentos, foi modelado o diagrama relacional da base de dados.

Com o diagrama relacional pronto e validado, foi feito o mapeamento para o diagrama entidade-relacionamento, o qual apresenta uma visão mais sólida do banco de dados no ambiente real. O diagrama entidade-relacionamento desenvolvido está disposto na figura 7.

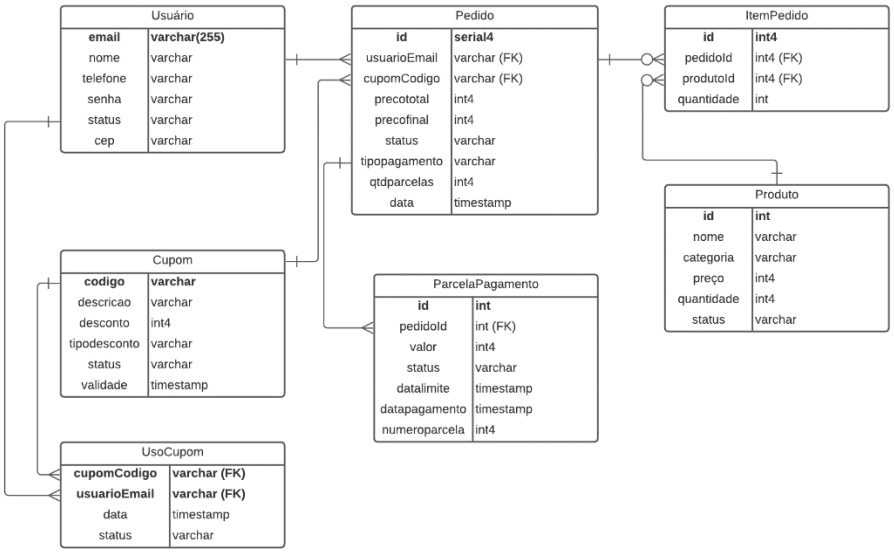

Figura 6 - Diagrama entidade-relacionamento da base de dados da aplicação desenvolvida. Fonte: Autor.

Por fim, foi feito a escolha do DBMS (Database Management System), onde foi optado pelo DBMS de código livre PostgreSQL.

3.3.3. Implantação e Conteinerização do Banco de Dados

A implantação do banco de dados foi realizada por meio de containers Docker. O Docker é uma tecnologia de conteinerização que permite o isolamento de diferentes processos dentro de uma máquina.

Para garantir o reuso do banco de dados para usuários que irão interagir com o sistema, foi utilizado o Docker Compose para definir as características da imagem do container. O arquivo DockerCompose com os comandos de inicialização do banco de dados pode ser visualizado no script 1.

```
postgres:
  container_name: postgres
  restart: always
  image: postgres:12.9-alpine
  volumes:
    - ./data:/var/lib/postgresql/data
  ports:
    - 5432:5432
  environment:
    - POSTGRES_PASSWORD=password
```

Script 1 – Comandos do arquivo docker-compose.yml para inicialização do container contendo o banco de dados PostgreSQL.

Ainda considerando a reusabilidade do banco de dados, foi implantado uma configuração de retenção do volume do container, de maneira com que os dados do banco de dados não sejam perdidos após a parada do processo do banco de dados.

Para gerenciar o banco de dados de maneira fácil e visual, foi utilizado a aplicação Desktop DBeaver, que utiliza a interface de programação de aplicativo JDBC para interagir com o banco de dados criado.

3.3.4. Desenvolvimento da Aplicação

A aplicação, como descrito anteriormente, foi desenvolvida utilizando o stack de tecnologia Node.js e express. Foi utilizado um padrão de arquitetura monolítico, onde os módulos interagem entre si através de chamadas de funções.

Na arquitetura desenvolvida, é realizada a divisão da aplicação entre a API e os controllers. A API é a interface direta do cliente com a aplicação, enquanto os controllers são responsáveis por tratar a lógica da aplicação e realizar as chamadas para o banco de dados.

Para a conexão com o banco de dados, foi optado pela tecnologia Sequelize, um ORM (Object-Relational Mapping) que permite a comunicação com o banco de dados através de um paradigma orientado a objetos.

O diagrama contendo o fluxo de informações da aplicação pode ser visualizado na figura 8.

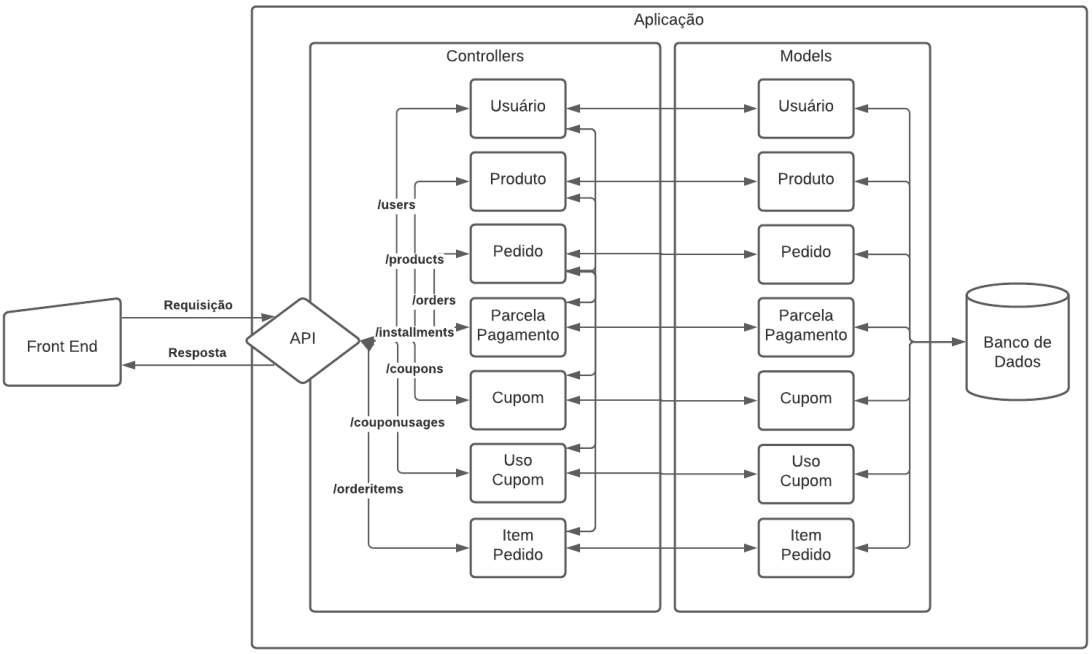

Figura 7 - Fluxo de Informações da Aplicação monolítica desenvolvida. Fonte: Autor.

Durante o desenvolvimento da aplicação, fez-se necessário o uso de diversos pacotes de código aberto para que se pudesse realizar funções de maneira mais simples. Os principais pacotes utilizados e suas descrições são:

- Express.js: Framework para desenvolvimento de servidores de aplicações web.
- Joi: Validador de parâmetros e esquemas. Utilizado para validar os parâmetros enviados nas requisições do cliente.
- Date-fns: Biblioteca com diversas funções de operação e formatação de datas. Utilizado para popular e comparar datas na aplicação.
- Lodash: Biblioteca de funções de operação de *arrays* e objetos. Utilizado para transformação de objetos na aplicação.

- Md5: Biblioteca que contém funções de *hash* utilizando o algoritmo md5. Utilizado para criptografar a senha dos usuários cadastrados.

- Sequelizer: ORM para conexão com o banco de dados. Utilizado para definição de modelos, buscas e transações no banco de dados.

- Jest: Framework de testes paralelos. Utilizado no desenvolvimento de testes unitários e de integração do sistema.

- Axios: Biblioteca para realização de requisições em APIs externas. Utilizado no desenvolvimento de testes para realizar requisições à API da aplicação.

3.3.5. Desenvolvimento dos Testes

Para a etapa de desenvolvimento de testes, foi considerado que os testes deverão funcionar também para uma arquitetura de microserviços. Por conta disso, os testes foram realizados através da biblioteca Axios, que permite uma chamada de API para um endereço especificado. Dessa maneira, é possível utilizar os mesmos testes para uma arquitetura diferente, sem necessidade de alterar os códigos desenvolvidos nos arquivos de testes.

Os testes foram desenvolvidos de maneira com que testem as funcionalidades descritas nos itens da seção 3.2. retornem os resultados de maneira correta. Antes do início dos testes de integração, é realizado a criação de itens chaves no banco de dados, que serão utilizados para realizar operações de integração. As funcionalidades esperadas para cada teste estão descritas na seção 3.3.5.1.

Para realizar os testes, é necessário utilizar o comando *npm run test. Ao* realizar o teste com o sistema funcionando corretamente, espera-se o retorno como ilustrado na figura 9.

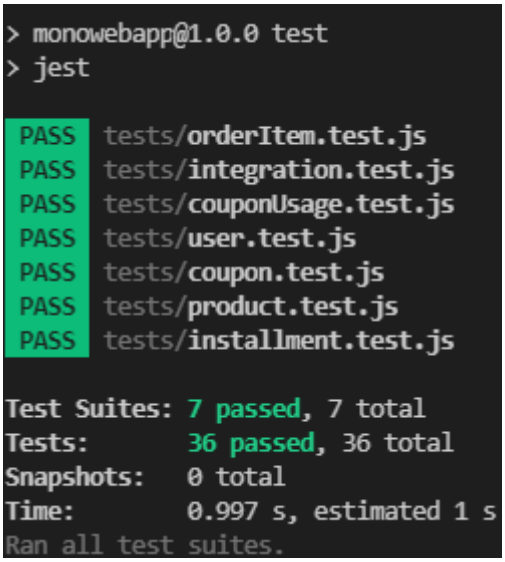

Figura 8 – Sucesso no retorno dos testes da aplicação. Fonte: Autor.

3.4. Resultados Obtidos

3.4.1. Considerações Iniciais

Após o aluno concluir o processo de migração da aplicação, é necessário que haja uma avaliação da metodologia adotada e também da arquitetura da aplicação final. Antes de tudo, é necessário que, após a migração completa, a aplicação na arquitetura de microserviços obtenha sucesso em todos os testes previamente desenvolvidos.

Espera-se que o aluno desenvolva um relatório de como o problema de migração foi abordado, de maneira que o avaliador possa avaliar não só o sistema final, mas também as decisões do processo de migração.

Por conta de haver diversos métodos de migração e diferentes modos de aplicar a arquitetura de microserviços, foi escolhido uma série de itens que avaliam de forma geral se a arquitetura apresentada corresponde a uma arquitetura de microserviços.

3.4.2. Avaliação da Arquitetura do Sistema

Para realizar a avaliação do sistema migrado, é necessário levar em consideração os principais pontos que definem uma arquitetura de microserviços. Como descrito no capítulo 2 deste trabalho, há dois pontos que definem uma arquitetura como microserviços:

- Definição de um protocolo para comunicação entre os microserviços (REST, Filas, etc.).

- Fontes de dados única para cada microsserviço – qualquer necessidade de obter dados de outro microsserviço, deve ser realizado uma comunicação através do protocolo pré-definido.

Caso a arquitetura do sistema apresentado pelo aluno não cumpra os requisitos acima, deve haver um desconto considerável na avaliação final do projeto, visto que são requisitos necessários para que se possa definir a arquitetura desejada.

É importante que o aluno descreva com suas próprias palavras o motivo da escolha do método de comunicação entre os microserviços, assim como suas vantagens e desvantagens.

Independente do padrão de microserviços adotado no projeto, deve ser necessário que os módulos do sistema original tenham uma divisão adequada para microserviços.

A avaliação da componentização do sistema deve tomar como fundamento o requerimento da divisão dos componentes pela lógica das diferentes capacidades de negócio [7].

Um exemplo de uma divisão de componentes baseado na capacidade de negócios pode ser visualizado na tabela 1.

Tabela 1 - Exemplo de divisão de componentes baseado na capacidade de negócios.

Microserviço	Módulos Utilizados	Funcionalidades

Usuário	Usuário	Cadastro, Autenticação, Recuperações, Atualizações e Remoções de usuários.
Pedido	Pedido, Item-Pedido	Operações de Criação, Atualização e Remoção de pedidos e itens do pedido.
Pagamento	Parcela-Pagamento	Operações relacionadas ao pagamento das parcelas de um pedido.
Cupom	Cupom, Uso-Cupom	Operações relacionadas ao uso de cupons em um pedido.
Produto	Produto	Operações de Criação, Atualização e Remoção de produtos do estoque.

É importante observar que a tabela 1 apresenta somente um exemplo de divisão dos componentes. É necessário que o avaliador possua um conhecimento das diferentes possibilidades para que seja possível avaliar de forma satisfatória o sistema migrado.

Ainda dentro da componentização, deve ser avaliada a efetividade do aluno em implantar os microserviços através de múltiplos processos isolados. Para atingir essa funcionalidade, espera-se que haja uma tecnologia de conteinerização ou virtualização, onde cada microserviço é implantado em um container ou em uma máquina virtual.

3.4.3. Avaliação da Metodologia de Migração

Como descrito no capítulo 2, espera-se que haja desafios ao realizar o processo de migração do sistema atual para o sistema com uma arquitetura de microserviços. É necessário que o aluno descreva, em seu relatório, os seguintes processos:

1. Qual foi o primeiro passo abordado para iniciar a migração do sistema?
2. Como foi realizado e qual tecnologia foi utilizada para realizar a migração do banco de dados único para múltiplos bancos?

3. Qual foram as alterações necessárias para que o sistema pudesse ser utilizado em um sistema conteinerizado ou virtualizado?

4. Quais foram os métodos utilizados para garantir que o sistema continuasse funcionando da mesma maneira?

As descrições dos processos devem ser levadas em conta para que o avaliador possa verificar se houve uma metodologia de migração coerente com as melhores práticas. Mesmo com diferentes possibilidades de realizar os processos, é possível avaliar a qualidade da migração com base nos esforços alocados para cada um dos itens descritos.

Para o item 1, espera-se que o aluno descreva com detalhes o processo de identificação das capacidades de negócio do sistema original, de maneira com que haja uma primeira visão de como seria possível realizar a divisão correta dos módulos para microserviços.

No item 2, é necessária uma explicação detalhada da identificação da tecnologia de banco de dados original, e também do estudo do relacionamento entre as tabelas. É também esperado o uso de uma tecnologia para auxiliar no processo de migração de dados e tabelas.

Para o item 3, é esperado uma descrição do processo de criação de imagens para cada microserviço criado, assim como as tecnologias utilizadas para realizar a implantação das imagens.

No último item, é necessário a descrição de como o serviço pode ser inicializado, de maneira com que todos os testes já escritos continuem funcionando normalmente.

3.5. Dificuldades e Limitações

Como descrito no capítulo 2, diversas são as maneiras de realizar uma implementação de uma arquitetura de microserviços. O aluno, ao realizar o processo de migração do sistema, poderá se deparar com diversas opções para que realizar os processos de implantação da aplicação na nova arquitetura. Por isso, há uma dificuldade em garantir de o avaliador possuir um conhecimento razoável das diferentes metodologias e padrões aceitáveis para uma

arquitetura de microserviços, de maneira com que a performance do aluno possa ser avaliada de forma satisfatória.

Além disso, é também necessário observar que o sistema monolítico desenvolvido é uma aplicação simples com mínimos processos necessários para que faça sentido em realizar uma migração para arquitetura de microserviços. Por conta disso, há uma limitação no que se diz à simulação de uma aplicação real, onde provavelmente existiria uma maior complexidade e uma quantidade muito maior de módulos.

Para trabalhos futuros, uma avaliação mais específica do que poderia ser aceito, para cada padrão de arquitetura de microserviços, traria benefícios que diminuiriam a necessidade de o avaliador possuir um conhecimento profundo de cada padrão. Também, um maior aprofundamento no desenvolvimento da aplicação seria favorável para que o sistema se aproximasse mais de um sistema real, o que traria uma maior dificuldade, e maior aprendizado, na realização do projeto pelo aluno.

3.6. Considerações Finais

A realização de um processo de migração de uma arquitetura monolítica para uma arquitetura de microserviços é um desafio cada vez mais presente no mundo do desenvolvimento de software.

A aplicação base desenvolvida exige que o aluno realize não só um desenvolvimento, mas também exige um estudo de como uma arquitetura de microserviços deve se comportar, tanto no âmbito de código, quanto na arquitetura de dados e implantações.

É esperado do aluno, portanto, um aprendizado prático do desenvolvimento de arquiteturas de microserviços, que será de grande importância caso seja deparado com um caso real em sua vida profissional.

CAPÍTULO 4: CONCLUSÃO

4.1. Contribuições

O sistema desenvolvido, assim como a metodologia de avaliação, possui como objetivo contribuir para o aprendizado de microserviços como um todo. Não só o aluno, mas o avaliador que fizer uso da plataforma também será bastante beneficiado com um aprendizado mais profundo das diferentes metodologias de migração e implantação de microserviços. Também, por conta de o código desenvolvido ser livre, é também esperado uma contribuição para o mundo Open-Source, que poderá utilizar a aplicação para referências futuras.

4.2. Considerações sobre o Curso de Graduação

Várias disciplinas do curso de Engenharia de Computação da EESC/ICMC contribuíram significativamente para o desenvolvimento do projeto. Em especial, a disciplina de “Engenharia de Software” e “Sistemas Distribuídos” foram essenciais para que se pudesse planejar o trabalho com bons conhecimentos de programação, planejamento de software e arquitetura de microserviços.

Apesar da quantidade de disciplinas voltadas para a Engenharia de Software serem limitadas, o curso de graduação de Engenharia de Computação, como um todo, contribuiu fortemente para o bom desenvolvimento do projeto. A grande cobrança dos professores e a dificuldade de algumas disciplinas, foram essenciais para o crescimento pessoal, acadêmico e profissional durante o andamento da graduação.

REFERÊNCIAS

[1] BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P.; TAMBURRI, D. A.; LYNN, T. (2018). **Microservices migration patterns. Software: Practice and Experience**. doi:10.1002/spe.2608. Acesso em 06 Nov. 2021.

[2] DRAGONI, Nicola, GIALLORENZO Saverio; LAFUENTE Alberto Lluch; MAZZARA, Manuel; MONTESI Fabrizio; MUSTAFIN, Ruslan; SAFINA , Larisa. (2017). **Microservices: Yesterday, Today, and Tomorrow. Present and Ulterior Software Engineering,** 195–216. doi:10.1007/978-3-319-67425-4_12. Disponível em https://link.springer.com/chapter/10.1007/978-3-319-67425-4_12. Acesso em 5 Set. 2021.

[3] ERL, T. SOA: **Principles of Service Design**. 1. ed. Prentice Hall; 1st edition (July 20, 2007).

[4] FRYE, Brent. **8 Steps for Migrating Existing Applications to Microservices. Carnegie Mellon University**. Disponível em https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/. Acesso em 06 Nov. 2021.

[5] LEWIS, James; FOWLER, Martin. 2014 **Microservices, a definition of this new architectural term.** Disponivel em: https://martinfowler.com/articles/microservices.html Acesso em 16 Nov. 2021.

[6] MAZZARA, M., DRAGONI, N., BUCCHIARONE, A., GIARETTA, A., LARSEN, S. T., & DUSTDAR, S. (2018). **Microservices: Migration of a Mission Critical System. IEEE Transactions on Services Computing**, 1–1. doi:10.1109/tsc.2018.2889087. Acesso em 06 Nov. 2021.

[7] PERREY, R.; LYCETT, M. (2003). **Service-oriented architecture**. doi:10.1109/saintw.2003.121013. Disponível em https://ieeexplore.ieee.org/abstract/document/1210138. Acesso em 5 Set. 2021.

[8] RICHARDSON, C.: (2014). **Microservices architecture.** Disponível em http://microservices.io/. Acesso em 04 Nov 2021.

[9] SHADIJA, Dharmendra; REZAI, Mo; HILL, Richard. (2017). **Towards an Understanding of Microservices.** Disponível em https://www.researchgate.net/publication/319952918_owards_an_Understanding_of_Microservices. Acesso em 13 Set. 2021.

[10] TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus. (2017). **Processes, Motivations and Issues for Migrating to Microservices Architectures: An Empirical Investigation.** IEEE Cloud Computing. 4. 10.1109/MCC.2017.4250931. Disponível em https://www.researchgate.net/publication/319187656_Processes_Motivations_and_Issues_for_Migrating_to_Microservices_Architectures_An_Empirical_Investigation. Acesso em 13 Set. 2021.

[11] TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus. (2018) **Architectural Patterns for Microservices: A Systematic Mapping Study.** 10.5220/0006798302210232. Acesso em https://www.semanticscholar.org/paper/Architectural-Patterns-for-Microservices%3A-A-Mapping-Taibi-Lenarduzzi/f6e88823482de1729584acbfb450d4502f4d393d. Acesso em 03 Out. 2021.

[12] VRESK, T; ČAVRAK I. (2016) **Architecture of an interoperable IoT platform based on microservices**. 10.1109/MIPRO.2016.7522321. Acesso em 04 Nov. 2021.

[13] WIRFS-BROCK, A; EICH, B. (2020) **Javascript: the first 20 years**. 10.1145/3386327. Acesso em 05 Nov. 2021.