UNIVERSIDADE FEDERAL DO MARANHÃO

Curso de Bacharelado em Ciência da Computação

Erickson Bruno Pereira Costa

Uma Arquitetura de Microsserviços aplicada ao Ambiente Virtual de Aprendizagem GReAT

São Luís - MA

2020

Erickson Bruno Pereira Costa

Uma Arquitetura de Microsserviços aplicada ao Ambiente Virtual de Aprendizagem GReAT

Trabalho de Conclusão de Curso Monografia apresentada a Universidade Federal do Maranhão (UFMA) para a obtenção do grau de Bacharel em Ciência da Computação.

Curso de Bacharelado em Ciência da Computação

Universidade Federal do Maranhão

Orientador: Alana Oliveira Meireles Teixeira

São Luís - MA

2020

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Pereira Costa, Erickson Bruno.
Uma Arquitetura de Microsserviços aplicada aoAmbiente Virtual de Aprendizagem GReAT / Erickson Bruno Pereira Costa. - 2021.
47 f.

Orientador(a): Alana Oliveira Meireles Teixeira.
Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís - MA, 2021.

1. API. 2. Arquitetura. 3. GReAT. 4. Microsserviços. I. Meireles Teixeira, Alana Oliveira. II. Título.

Erickson Bruno Pereira Costa

Uma Arquitetura de Microsserviços aplicada ao Ambiente Virtual de Aprendizagem GReAT

Trabalho de Conclusão de Curso Monografia apresentada a Universidade Federal do Maranhão (UFMA) para a obtenção do grau de Bacharel em Ciência da Computação.

Monografia aprovada em São Luís - MA, 04 de Janeiro de 2021:

Alana Oliveira Meireles Teixeira
Orientadora
Universidade Federal do Maranhão

Prof. Dr. Mário Meireles Teixeira
Universidade Federal do Maranhão
Professor

Prof. Me. Carlos Eduardo Portela Serra de Castro
Professor
Universidade Federal do Maranhão

São Luís - MA
2020

Dedico este trabalho a todos os que me ajudaram ao longo desta caminhada.

Agradecimentos

Agradeço primeiramente a Deus pela saúde e força para continuar nesta jornada.

Aos meus pais que sempre me deram o apoio necessário para continuar caminhando em direção a conclusão desta etapa.

Aos meu amigos que me ajudaram diversas vezes ao longo dos anos.

Aos meus professores, pelo empenho em transmitir da melhor forma seus conhecimentos.

A minha Orientadora Alana Oliveira pela excelente orientação e ajuda, sem a qual eu não poderia concluir esta etapa.

"Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes."

(Marthin Luther King)

Resumo

É crescente o uso de aplicações computacionais no cotidiano das pessoas, podendo ser usadas tanto para fins financeiros, como em bancos e compras online, quanto para entretenimento e até mesmo para fins educacionais. Desta forma, desenvolver aplicações web que suportem uma quantidade cada vez maior de acesso simultâneo tem sido uns dos grandes desafios desse mundo cada vez mais virtual. Atrelado a isso, o desenvolvimento de arquiteturas mais flexíveis e escaláveis tem ganhado destaque nos últimos anos. As arquitetura monolíticas, ditas tradicionais, costumam ter um elevado custo de manutenção quando utilizado para suportar um grande volume de dados, pois requerem uma escalabilidade vertical, sendo necessário aumentar a capacidade física do servidor, adicionando mais memória e mais processadores, ou então replicando a aplicação completamente em outro servidor. Assim, este trabalho vem propor uma arquitetura em microsserviços como uma solução de estruturação distribuída, escalável e de fácil integração aplicado ao ensino. Para tal, foi desenvolvida uma aplicação web que utiliza arquitetura de microsserviços para se comunicar com uma API Externa trocando e processando informações para definir estilos de aprendizagem de alunos da ferramenta GReAT. Ao final buscou-se analisar os resultados de forma a apontar os pontos positivos e negativos deste tipo de arquitetura e as formas de utilizá-la

Palavras-chave: Arquitetura, Microsserviços, GReAT, API.

Abstract

There is a growing use of computational applications in people's daily lives, which can be used both for financial purposes, as in banking and online shopping, as well as for entertainment and even for educational purposes. Thus, developing web applications that support an increasing amount of simultaneous access has been one of the great challenges of this increasingly virtual world. Coupled with this, the development of more flexible and scalable architectures has gained prominence in recent years. Monolithic architecture, so-called traditional, usually has a high maintenance cost when used to support a large volume of data, as they require vertical scalability, being necessary to increase the physical capacity of the server, adding more memory and more processors, or replicating the application completely on another server. Thus, this work proposes a microservice architecture as a distributed, scalable and easily integrated structuring solution applied to teaching. To this end, a web application was developed that uses microservices architecture to communicate with an External API, exchanging and processing information to define learning styles for students using the GReAT tool. At the end, we tried to analyze the results in order to point out the positive and negative points of this type of architecture and the ways to use it.

Keywords: Architecture, Microservices, GReAT, API

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
REST	*Representational State Transfer*
URL	*Uniform Resource Locator*
HTTP	*Hypertext Transfer Protocol*
GReAT	*Game Ready Authoring Tool*
JSON	*JavaScript Object Notation*
XML	*Extensible Markup Language*
TDD	*Test-driven development*
VM	*Virtual Machine*
UFMA	*Universidade Federal do Maranhão*
HTML	*HyperText Markup Language*
CSS	*Cascading Style Sheets*
ODM	*Object Data Model*
NoSQL	*Not Only SQL*
SQL	*Structured Query Language*

Sumário

1 Introdução

1.1 Contextualização

Ao longo dos anos, as aplicações web tem ganhado papel importante no cotidiano da população mundial. Sites, comércio eletrônico, redes sociais, entre outros, têm feito cada vez mais parte da vida das pessoas. Pesquisas recentes, (MEIRELLES, 2020), apontam que a difusão dos smartphones inclusive contribuiu muito nesse sentido, pois em 2020 no Brasil existem 424 milhões de dispositivos digitais em uso , além de 134 milhões de usuários de internet (INFORMAÇÃO; PONTO, 2020).

Um volume desse traz um desafio maior aos desenvolvedores, pois precisam desenvolver aplicações web que suportem uma quantidade cada vez maior de acesso simultâneo. Atrelado a isso, o desenvolvimento de arquiteturas mais flexíveis, escaláveis tem ganhado destaque nos últimos anos.

A primeira arquitetura utilizada para desenvolver aplicações é a chamada arquitetura monolítica. Nesse tipo de arquitetura um único executável contém toda aplicação, desde a interface do usuário, o processamento da aplicação, até a comunicação com o banco de dados. O exemplo mais comum que vem à mente ao discutir monólitos é um sistema no qual todo o código é implantado como um único processo. Pode-se ter várias instâncias desse processo por motivos de robustez ou escalabilidade, mas fundamentalmente todo o código é compactado em um único processo (NEWMAN, 2019).

Embora haja algum tipo de modularização interna, ou seja, há divisões bem definidas dentro das aplicações, esse tipo de arquitetura costuma ter um custo elevado de manutenção quando utilizado para suportar um grande volume de dados, pois, ao precisar de mais poder de processamento é necessário aumentar a capacidade física do servidor, adicionando mais memória e mais processadores, ou então replicando a aplicação completamente em outro servidor.

Como alternativa a arquitetura citada anteriormente, em 2011 foi proposta por (FOWLER, 2014) uma nova abordagem, uma arquitetura em que seus componentes não fossem mais acoplados e sim dispersos, porém bem integrados através de diversos protocolos de comunicação.

Este trabalho visa estabelecer um estudo dessa arquitetura bem como prover uma aplicação utilizando os fundamentos da arquitetura de microsserviços, capaz de comunicar-se com outras APIs Externas.

1.2 Justificativa

Devido a dificuldade em se manter sistemas complexos desenvolvidos com uma arquitetura monolítica, essa pesquisa se justifica através da análise de arquiteturas que venham facilitar a integração de diferentes módulos e funcionalidades voltados a plataformas de ensino.

1.3 Objetivo

Nesta seção serão apresentados os objetivos, geral e específicos, do trabalho a ser desenvolvido nesta monografia.

1.3.1 Objetivo Geral

O presente trabalho tem como objetivo geral propor uma arquitetura em microsserviços como uma solução de estruturação distribuída, escalável e de fácil integração com ambiente virtual de aprendizagem GReAT.

1.3.2 Objetivos Específicos

- Conceituar Microsserviços e suas aplicações.
- Desenvolver uma proposta de arquitetura aplicada ao domínio em questão.
- Integrar a arquitetura proposta com o ambiente virtual de aprendizagem GReAT.

1.4 Organização e Estrutura

O trabalho aqui proposto possui uma estrutura dividida em capítulos.

- O Capítulo 1 nos traz uma introdução acerca do objeto de estudo deste trabalho.
- O Capítulo 2 contém uma descrição de alguns trabalhos relacionados ao tema principal deste trabalho e as usas diferenças principais.
- O Capítulo 3 contém uma explicação dos conceitos necessários para se compreender melhor o objeto de estudo.
- O Capítulo 4 é demonstrado a arquitetura adotada.
- O Capítulo 5 e apresentado uma análise acerca dos resultados obtidos ao final da pesquisa e desenvolvimento.
- O capítulo 6 temos uma conclusão sobre o tema deste trabalho.

2 Trabalhos Relacionados

Nesta seção nosso objetivo é explorar e entender através de pesquisa em trabalhos relacionados, quais os benefícios em se utilizar este tipo de arquitetura em detrimento da arquitetura já existente no meio de desenvolvimento web, quais desafios outros pesquisadores já passaram e extrair alguma premissa válida de suas pesquisas para os fins deste trabalho.

2.1 Comparação de Custos de Infraestrutura

O artigo (VILLAMIZAR et al., 2016) fez um estudo que comparava os custos de infraestrutura da Web usando AWS Lambda e Arquiteturas Monolíticas e de Microsserviço, como métrica de comparação os autores em questão optaram por fazer testes de estresse utilizando cada uma das arquiteturas.

Os testes consistiam no envio de uma massa solicitações e obtenção das resposta estabelecendo um limite mínimo de tempo para tal em milissegundos. Os testes de estresse foram executados com o objetivo de identificar o número máximo de solicitações suportadas pelas arquiteturas até que as mesmas apresentassem erro ou performance acima do tempo limite especificado.

Os resultados mostraram que a arquitetura de microserviços suportando o mesmo número de solicitações tinha um custo menor se comparado a arquitetura monolítica, além disso, suportava mais solicitações por minuto que a outra arquitetura. Isso nos diz que a arquitetura de microsserviços se mostra promissora no sentido de trazer um melhor aproveitamento dos recursos computacionais, e um menor custo benefício ao ser utilizada.

Este trabalho foi feito com o intuito de estabelecer uma alternativa para a arquitetura utilizada atualmente no ambiente de ensino GReAT, seja como arquitetura principal ou apenas aplicações auxiliares. Os resultados deste experimento podem nos mostrar quais as possibilidades podem ser utilizadas com intenção de garantir um melhor aproveitamento dos recursos sem descaracterizar o objetivo ao qual o ambiente de ensino se propõe.

Diferente do trabalho apresentado anteriormente nosso propósito aqui não é comparar os diferentes tipos de arquiteturas e estabelecer uma métrica de qualidade entre elas e sim, estabelecer um estudo sobre a arquitetura e ter uma visão de como aplicações com arquitetura baseada em microserviços podem ser utilizadas, seja em conjunto a aplicações que utilizam outros tipos de arquitetura, ou como arquitetura principal.

2.2 GReAT (Game Ready Authoring Tool)

GReAT é um ambiente de autoria de jogos sérios, criado em 2014 por pesquisadores da Universidade Federal do Maranhão (UFMA) e, em sua versão inicial, era composta por quatro módulos: Authoring, Catalog, Game Center e Play (OLIVEIRA; NETO; TEIXEIRA, 2014).

Tendo sido desenvolvida para atender a necessidade do professor de ter mais ferramentas que auxiliem no ensino do conteúdo acadêmico, e utilizando métodos de gamificação no ensino para ajudar no aprendizado dos alunos. Assim, a ferramenta GReAT se propõem ajudar os professores, de forma simples, na criação de jogos que proporcionem aos alunos uma experiência de ensino mais agradável e que tenha uma boa eficácia na transmissão do conhecimento. Alguns requisitos fundamentais observados para o cumprimento dos objetivo propostos na ferramenta são: facilidade de uso, desafio, visual atrativo, divisão em níveis, ranqueamento e compartilhamento de resultados.

Para produzir os jogos, foi desenvolvido um ciclo que vai desde a ideia inicial, concepção do jogo, publicação, obtenção dos resultados e por fim a divulgação dos mesmos. Inicialmente tem-se um módulo chamado de Authoring Tool responsável por obter os dados referentes ao jogo e pela construção. Em seguida, a módulo Catalog, responsável pela exibição dos jogos publicados no módulo anterior, é nesse módulo que os alunos podem escolher o jogo que querem jogar. Logo após escolher o jogo, o aluno é direcionado ao módulo chamado Play, é aqui que ele é exibido. E após jogar, suas estatísticas são coletadas e exibidas em outro módulo chamado Game Center.

Em sua versão atual, outros módulos foram adicionados à ferramenta, como o Classrom que é uma sala de aula virtual onde os professores interagem com os alunos, enviando-lhes conteúdo para estudo, tarefas, avisos, além dos jogos desenvolvidos no módulo de Autoria.

Posteriormente sentiu-se a necessidade de se ter um sistema que fornecesse um conteúdo que fosse melhor aproveitado pelo aluno, com base em seu estilo de aprendizagem (KOLB, 1984). A motivação para a criação de um sistema de recomendação de conteúdo com base em seu perfil foi exposta em (OLIVEIRA; TEIXEIRA; NETO, 2020). Onde, de acordo com os autores, dado aos professores uma maneira de identificar e compreender melhor os estilos de aprendizagem dos alunos, eles serão capazes de fornecer melhores experiências educacionais e guiar o aprendizado de cada aluno de forma mais eficiente, de acordo com seu perfil.

Para identificar o perfil de aprendizado dos alunos, criou-se um modulo chamado Learning Style, responsável por aplicar aos usuários um questionário proposto por Honey e Mumford (HONEY.; MUMFORD, 2001). Esse questionário é composto por 80 questões sendo 20 questões para cada estilo de aprendizagem, e cujas respostas eram sim ou não

para cada pergunta. Ao final, obtemos o estilo de aprendizagem dominante.

Com base no Perfil de aprendizagem, e utilizando mineração de dados os melhores conteúdos para cada estilo eram recomendados no modulo Catalog.

3 Fundamentação Teórica

Este capítulo apresenta os conceitos explorados para o desenvolvimento do estudo, bem como embasar o leitor as tecnologias utilizadas.

3.1 REST

O Hypertext Transfer Protocol (HTTP) é o principal protocolo de comunicação utilizado pelas aplicações web e, ao longo dos anos, vem sofrendo diversas atualizações. Uma delas foi sugerida por Roy Fielding (FIELDING; TAYLOR, 2000), que entre outras coisas propôs o uso de novos métodos semanticamente mais significativos para alguns tipos de requisições. Essa proposta permitiu o uso do protocolo HTTP de forma mais compreensível. Por exemplo, a requisição DELETE intuitivamente nos diz que estamos tentando deletar um item através dessa requisição.

REST é o acrônimo de Representational State Transfer e seu objetivo é estabelecer padrões fundamentais para a construção de aplicações Web bem definidas, permitindo então a comunicação com outras aplicações. Aplicações que utilizam o padrão REST são chamadas Aplicações RESTful. Em particular, as APIs permitem a criação de padrões e podem funcionar com tecnologias básicas conhecidas e bem testadas, como o HTTP (DOGLIO; DOGLIO; CORRIGAN, 2018).

O padrão REST estabelece o uso dos métodos GET, POST, PUT, DELETE e PATCH. O método GET estabelece requisições que têm o intuito de obter dados do servidor. Já o método POST tem a finalidade de enviar dados ao servidor para que seja adicionado o conteúdo no backend. O PUT é utilizado para atualizar, o DELETE para deletar e, por fim, o PATCH é um método semelhante ao PUT porém neste não há a necessidade de enviar todos os parâmetros, e sim apenas os que foram atualizar.

O REST estabelece que um código de status deve ser adicionado na resposta da requisição para indicar o resultado do processamento. Além disso, a aplicação pode retornar um resultado em diversos formatos como JSON, XML e texto por exemplo, sendo o mais comum o formato JSON.

3.2 Arquitetura Monolítica

A Arquitetura Monolítica (Figura 1) é um tipo de arquitetura onde um único executável contém toda aplicação, desde a interface do usuário, realiza o processamento da aplicação, até a comunicação com o banco de dados. É a primeira arquitetura utilizada

na criação de sistemas operacionais sendo utilizada amplamente até os dias de hoje.

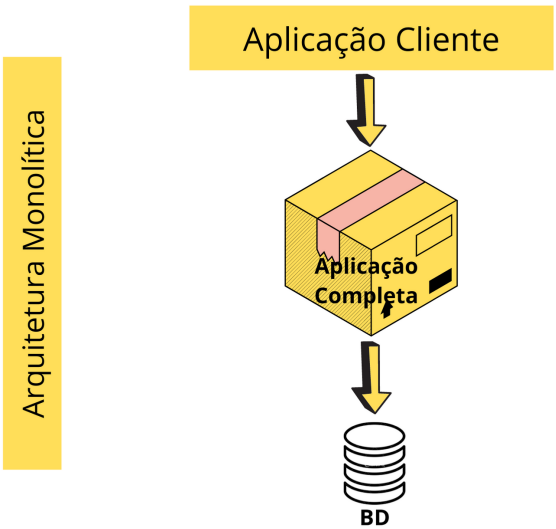
A maioria dos programas feitos com esse tipo de arquitetura possui modularização interna, permitindo que a aplicação seja dividida em vários módulos internamente. Além disso, possui uma boa definição de parâmetros entre suas estruturas, linguagens orientadas a objeto por exemplo reaproveitam partes comuns do código geralmente chamados de classes ou estruturas, evitando a duplicidade de código.

Boa parte dos programas são feitos sob medida, isso quer dizer que são desenvolvidos visando solucionar um problema específico, o que torna seu desenvolvimento mais fácil. Essa vantagem traz consigo uma desvantagem, por serem feitos sob medida, eles acabam tendo uma alta dependência ao objetivo inicial de seu desenvolvimento, tornando mais difícil modificar sua estrutura para atender a um novo objetivo. Adiciona-se a isso o fato de que quanto mais a aplicação cresce, mais difícil se torna a sua manutenção.

Devido a sua alta integrabilidade, um único erro em uma linha de código pode ocasionar uma quebra do sistema, tornando esse tipo de arquitetura pouco fiável. Felizmente existem diversas técnicas disponíveis para evitar esse tipo de problema, um exemplo disso é o desenvolvimento orientado a testes ou simplemente TDD (sigla do inglês, Test-driven development), que é um tipo de técnica programação que permite a criação de teste automatizados feito ao longo do processo de desenvolvimento, permitindo a descoberta de erros no sistema aumentando a qualidade do produto final.

A escalabilidade se torna também um problema, pelo fato da aplicação ser executada inteiramente em um único bloco. Caso um módulo interno necessite de mais recursos seria necessário aumentar os recursos de forma vertical (mais memória, ou capacidade de processamento) e se for o caso, replicar todo o sistema (horizontalmente). Isso demanda muito recurso, ocasionando um aumento de custo.

Figura 1 – Arquitetura Monolítica



Fonte: Elaborada pelo autor.

Apesar dos diversos problemas o uso desse tipo de arquitetura ainda é interessante uma vez que existem diversos tipo de técnicas, padrões de programação, frameworks desenvolvidos ao longo de anos voltados para o criação de aplicações monolíticas, e com bastante resultados práticos. Todavia, a utilização de uma arquitetura de microsserviços tem sido uma boa alternativa para o desenvolvimento de novas aplicações.

3.3 Arquitetura de Microsserviços

Arquitetura de Microsserviços é um tipo de arquitetura de software que consiste em uma aplicação composta em vários componentes mínimos, todos independentes uns dos outros, empenhados em solucionar um único problema.

A comunicação é feita utilizando protocolos leves como HTTP ou através de sistemas de mensageria como Apache Kafka[1], RabbitMQ[2], Amazon SQS[3], ActiveMQ[4] por exemplo. Essa comunicação garante a integração entre os diferentes serviços e com isso, a integridade do sistema como um todo.

Cada serviço pode ser desenvolvido, implantado e se for o caso, escalonado de forma independente, isso quer dizer que um serviço pode falhar ou funcionar sem que isso possa comprometer o funcionamento dos outros serviços.

Para ilustrar tomemos como exemplo um e-commerce que durante a black friday tem suas vendas normalmente alavancadas consumindo bastante recurso do servidor, ao utilizar uma arquitetura de microsserviços, pode-se ao invés de aumentar recursos do servidor (memoria, processador), apenas escalonar os serviços responsáveis por processar as vendas durante esse período e se for o caso, reduzir os recursos necessários para atender a outros microsserviços que não necessitem de tanto processamento nesse período. Utilizar essa arquitetura garante que todos os recursos serão bem utilizados, evitando assim desperdícios.

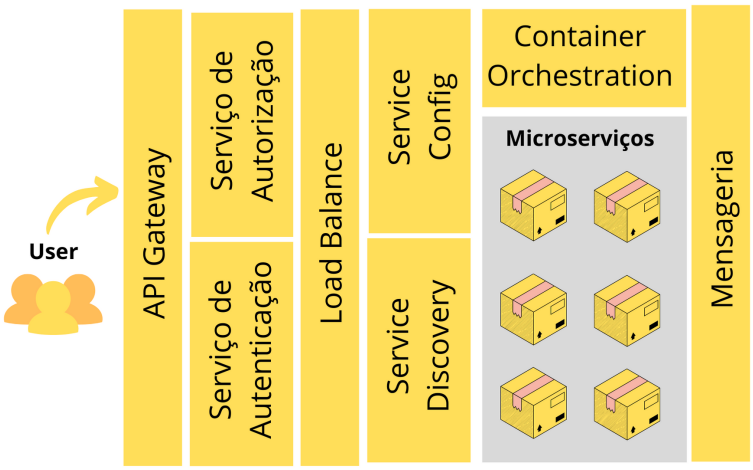
A arquitetura básica dos microsserviços é composta de várias etapas (Figura 2), cada uma delas tem seu grau de importância para o bom funcionamento do sistema como um todo, e serão descritas a seguir.

O API Gateway (Figura 3) é uma ferramenta que gerencia as aplicações. Tem a função de ser uma porta de entrada. Local em que todas as requisições dos usuários chegam e são redirecionadas aos seus respectivos serviços.

Os API Gateways proporcionam uma melhor usabilidade do sistema, uma vez que uma única solicitação, pode se dividir internamente em várias requisições internas a

[1] site: https://kafka.apache.org/
[2] site: https://www.rabbitmq.com/
[3] site: https://aws.amazon.com/pt/sqs/
[4] site: https://activemq.apache.org/

Figura 2 – Arquitetura Microsserviços

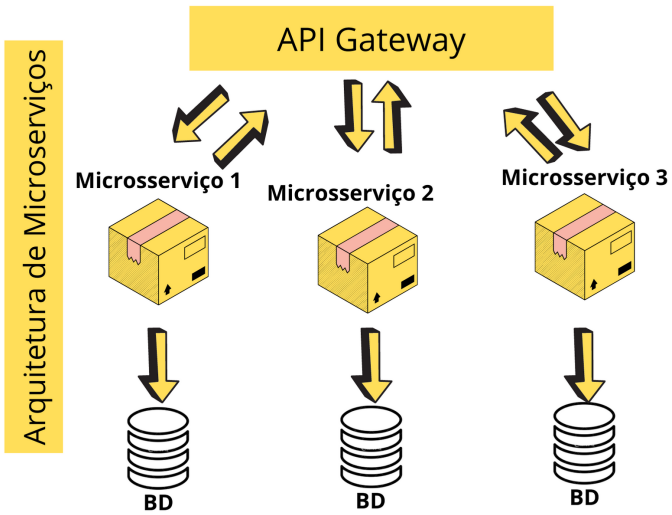


Fonte: Elaborada pelo autor.

diferentes microsserviços e depois retornando com o resultado da solicitação.

Os API Gateways modernas fornecem um recurso adicional crítico exigido por microsserviços: transformação e orquestração (NADAREISHVILI et al., 2016).Tudo isso sem o usuário sequer notar que está acontecendo. Além disso, seu uso traz consigo mais segurança, pois todas as chamadas para os serviços passam por um único ponto, ficando assim mais fácil de monitorar e analisar todas as requisições.

Figura 3 – API Gateway

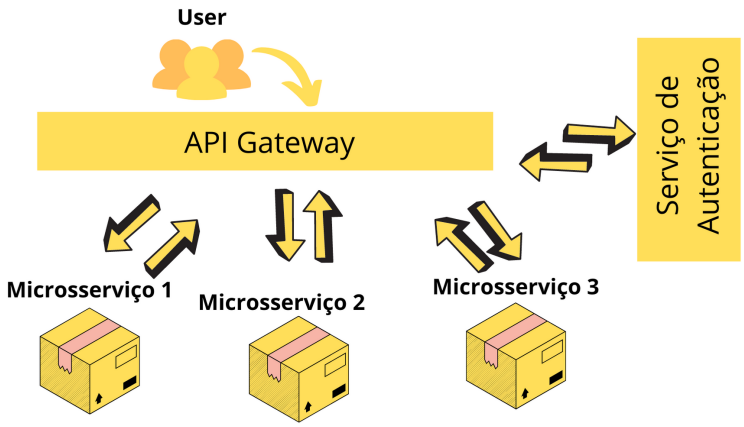


Fonte: Elaborada pelo autor.

Antes de redirecionar, porém, há ainda algumas etapas a serem cumpridas. Logo após fazer uma requisição para o API Gateway, ela faz uma verificação para saber se o usuário em questão está devidamente autenticado no sistema, para tal, o serviço de autenticação é chamado (Figura 4), caso o usuário não esteja logado, as credenciais serão

requisitadas e uma vez autenticado, as informações referentes ao usuário como nome e tempo de seção passa a ser monitorado e então poderá passar para a próxima etapa.

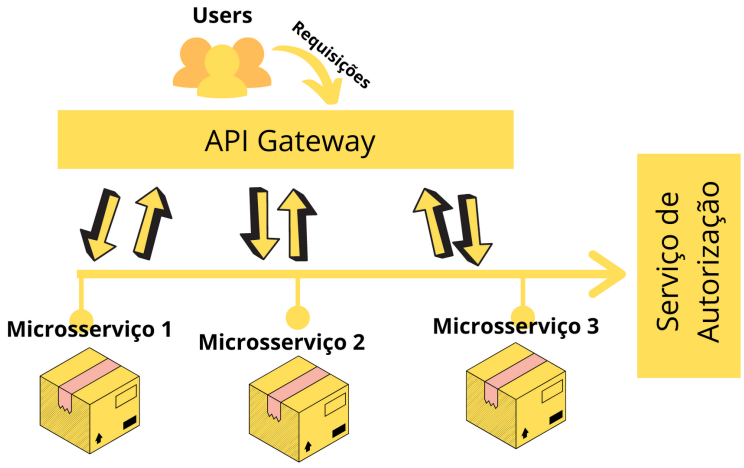
Figura 4 – Serviço de Autenticação



Fonte: Elaborada pelo autor.

Após passar pelo API Gateway e estar autenticado no sistema, o serviço de autorização (Figura 5) verifica se o usuário logado tem as devidas permissões para acessar aquele determinado serviço, caso tenha seu acesso será concedido.

Figura 5 – Serviço de Autorização

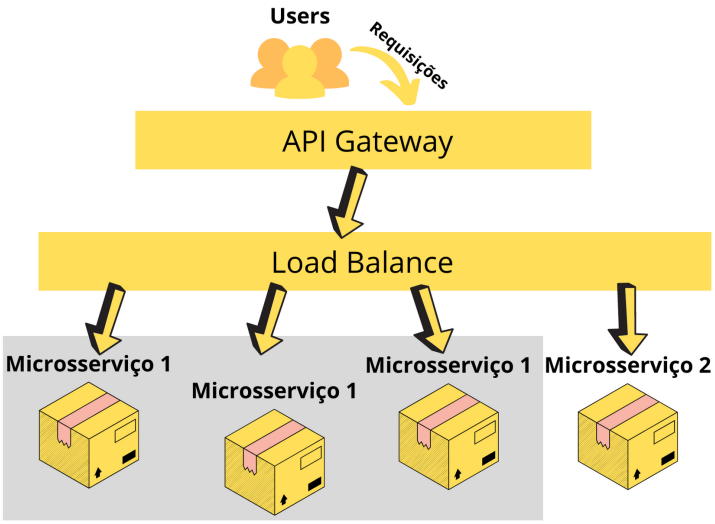


Fonte: Elaborada pelo autor.

Passadas as etapas de Autenticação e Autorização a requisição chega até o devido serviço. No contexto de microsserviços, o mesmo pode ser escalonado horizontalmente, podendo o mesmo microsserviço ter várias instâncias executando no mesmo ambiente. Para administrar a quantidade de requisições que cada instância recebe existe uma aplicação

chamada Load Balance (Figura 6), onde é balanceado a carga de requisições que cada instância recebe, diminuindo a ociosidade e reduzindo desperdícios ou sub-usos de recursos.

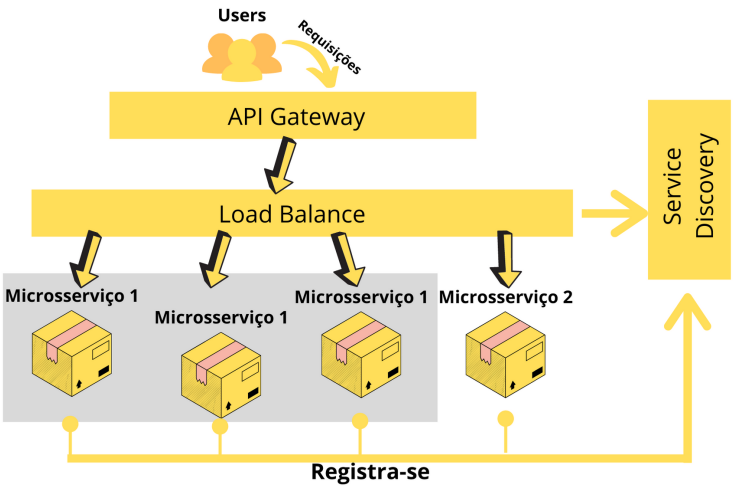
Figura 6 – Load Balance



Fonte: Elaborada pelo autor.

Para que o API Gateway saiba quais serviços estão em operação e com isso redirecionar para o local correto, é necessário que cada microsserviço informe seu endereço e a porta de uso. Isso pode ser feito de forma manual no momento da configuração do sistema, ou por meio dinâmico utilizando um Service Discovery (Figura 7). Em um contexto dinâmico, os endereços dos serviços podem mudar a qualquer momento, novas instâncias do mesmo microsserviço pode ser criada e com isso a rota para se chegar a um determinado microsserviço muda também.

Figura 7 – Service Discovery

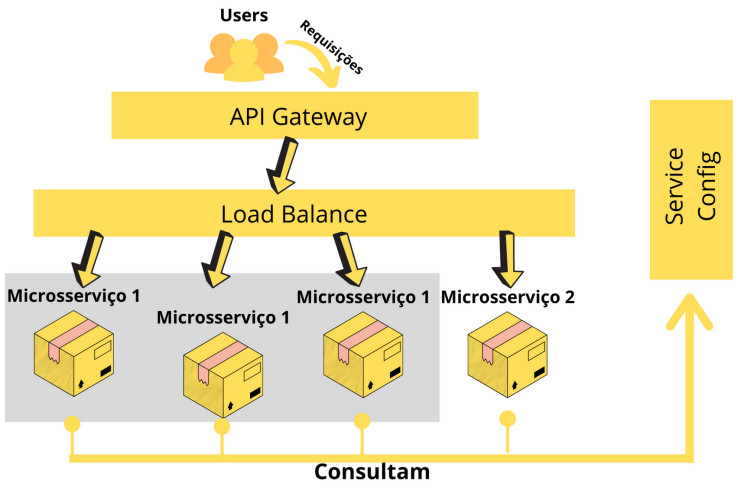


Fonte: Elaborada pelo autor.

O Service Discovery (Figura 7) é responsável por manter a lista de microsserviços ativos atualizadas, concentrando as rotas em um único lugar, isso facilita na hora de escalonar um determinado microsserviço. No momento que uma nova instância do serviço é ativada, registrando seu endereço atual no Service Discovery tornando-o visível para o Load Balance.

Ainda falando de agregadores temos também o Service Config (Figura 8), que é responsável por centralizar as configurações necessárias para o bom funcionamento do sistema como um todo, quando um microsserviço é instanciado, o mesmo consulta o Service Config para saber as configurações necessárias para conectar em seu respectivo banco. Nele é armazenado por exemplo as configurações com os bancos de dados e certificados públicos.

Figura 8 – Service Config.



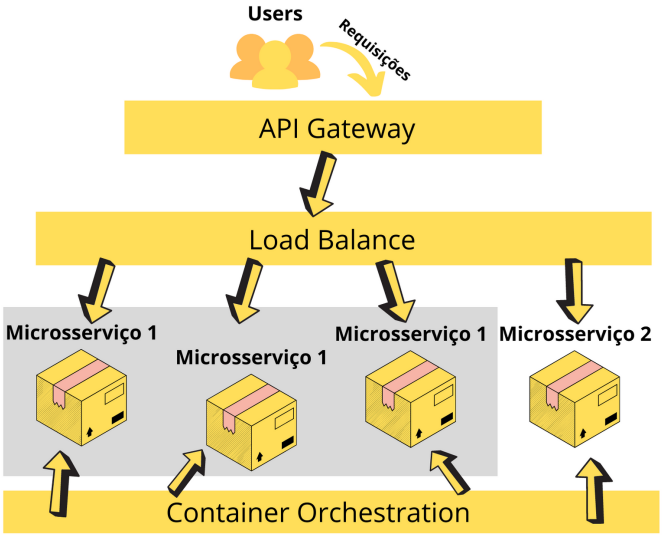
Fonte: Elaborada pelo autor.

Em ambientes de produção geralmente existem vários microsserviços e várias instâncias executando ao mesmo tempo, gerenciar esse sistema pode ser uma tarefa difícil, dependendo do seu tamanho e complexidade. Por exemplo, saber se um determinado serviço está sendo super utilizado ou subutilizado, e com isso escalonar ou reduzir suas instâncias.

Para gerenciar as instâncias dos serviços é necessário um Orquestrador de Contêiner (Figura 9), nele é possível estabelecer regras de gerenciamento dos serviços como por exemplo, instanciar novos serviços que estão sendo bastante requisitados, baixar o número de serviços que estão sendo pouco utilizados ou instanciar serviços que que por algum motivo caíram.

Como já dito, em arquiteturas baseadas em microsserviços, cada um cuida apenas de uma função, porém é muito comum um microsserviço comunicar-se com outro, para tal tarefa utiliza-se um serviço de mensageria.

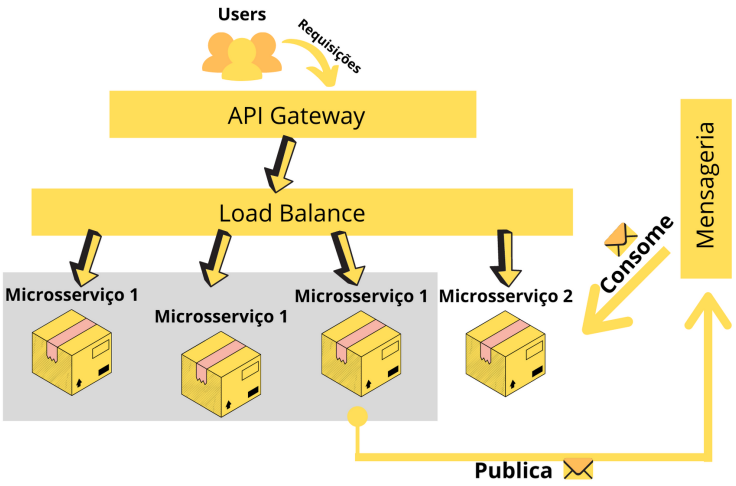
Figura 9 – Container Orchestration



Fonte: Elaborada pelo autor.

O Serviço de Mensageria (Figura 10), é responsável por centralizar o recebimento e sinalização de eventos, mensagens que um microsserviço manda a outro. É comum o uso de filas de mensagens onde um microsserviço publica em uma fila de outro microsserviço uma determinada mensagem e o outro microsserviço consome e processa essa mensagem enviada. O uso de mensageria faz mais sentido quando as mensagens são do tipo assíncronas, umas vez que as mensagens são publicadas em uma fila, ou seja só serão processadas quando as outras mensagens a sua frente forem consumidas, em um contexto de computação, esse tempo pode ser suficientemente longo para se esperar uma resposta "instantânea".

Figura 10 – Mensageria



Fonte: Elaborada pelo autor.

3.4 Conteinerização

Os contêineres são pacotes de software portáteis que detém apenas o suficiente para a execução da aplicação contida em si, como o código, bibliotecas e ferramentas de sistema. São frequentemente confundidas com Maquinas Virtuais (VM, pela sigla em inglês) responsáveis por simular um sistema operacional contendo além dos componentes normalmente contidos em um contêiner, como todo o código do sistema operacional simulado. Contêineres consomem menos recursos computacionais que uma VM pois compartilham recursos sem a necessidade de sustentar um sistema operacional inteiro, por esse motivo são normalmente muito mais leves que uma VM.

A premissa básica dos contêineres é que, cada um faça apenas uma única função, isso não só torna o processo de desenvolvimento mais prático, como maximiza a produtividade. Além disso, o uso dos contêineres torna bem mais eficiente o uso dos recursos computacionais. Para facilitar a escalabilidade e resiliência, muitas organizações agora executam aplicativos em ambientes nativos de nuvem usando contêineres e orquestração (RICE, 2020).

3.5 Tecnologias e Componentes

3.5.1 Linguagem de Programação

A linguagem de programação utilizada é o Javascript, por ser uma linguagem bastante conhecida e por possuir diversos frameworks que auxiliam no desenvolvimento da ferramenta que será descrita neste trabalho.

O Javascript, ou ECMAScript é uma linguagem de programação interpretada que utiliza scripts de alto nível, com tipagem fraca e com suporte a diversos paradigmas de programação. Juntamente com o HTML e CSS é umas das três principais linguagens de programação utilizadas para desenvolver aplicações web. Geralmente o Javascript é utilizado como linguagem base de diversos frameworks, que auxiliam na criação de aplicações de forma mais dinâmica, e assim trazendo um aumento de produtividade significativo.

Para auxiliar no desenvolvimento utilizando a linguagem descrita no parágrafo anterior, utilizou-se o Nodejs. O Nodejs é um ambiente de execução Javascript muito utilizado para criar aplicações do tipo server-side, aplicações node executam “sozinhas”, sem precisar de um browser para ser executado, isso porque o próprio Nodejs executa esse código como um servidor.

Do ponto de vista do servidor, o Nodejs possui muitos benefícios como, alta performance, uma vez que ele foi projetado para otimizar a escalabilidade das aplicações web. A portabilidade, pois possui suporte para vários tipos de sistemas operacionais, como

FreeBSD, Linux, Microsoft Windows por exemplo. Além disso, possui uma comunidade enorme, o que facilita na hora de tirar dúvidas acerca do desenvolvimento utilizando essa tecnologia.

Atrelado ao Nodejs temos o ExpressJs, um framework utilizado na criação de aplicações web para o Nodejs. Com ele podemos gerenciar requisições de verbos HTTP como Get e Post, integrar diversos tipos de motores de visualização (View Engines) e assim definir templates padrão e os dados a serem inseridos neles, definir configurações comuns das aplicações, como porta de conexão, a localização dos diferentes modelos usados para renderizar as respostas das requisições, além de trazer uma abstração para implementação de middlewares.

Para ajudar na construção do template utilizou-se o express-handlebars[5], que é um processador de templates, com ele podemos gerar páginas HTML dinamicamente, poupando esforço na hora de dar a manutenção para o código. O Handlebars utiliza HTML juntamente com a estrutura JSON, ou seja, a estrutura básica é escrita em HTML regular e junto são inseridos placeholders que são atributos utilizados para injetar informação quando for necessário. Outra vantagem em se utilizar essa tecnologia é o fato de um template padrão poder ser particionado, cada "partials" como é chamado, pode estar em um arquivo separado e ser utilizado da maneira que se achar melhor, isso traz dinamismo na construção do front-end.

Para trabalhar com Autenticação foi utilizado um middleware chamado Passport[6], que auxilia na segurança da aplicação. O Passport utiliza as chamadas "estratégias" para autenticar o usuário no sistema, essas estratégias nada mais são do que formas de se logar no sistema, seja utilizando o login do Facebook, Gmail, ou outra das mais de 500 formas diferentes. No nosso caso, para fins didáticos, estamos utilizando a estratégia local, que utiliza um usuário e senha definidos na aplicação.

Para a comunicação com o banco de dados foi utilizado o Mongoose[7], que é um ODM (Object Data Model), ou seja, uma espécie de interface para o banco de dados. O Mongoose cria e utiliza modelos de referência que interagem com o banco de dados MongoDb facilitando a obtenção e/ou persistência dos dados.

Para a persistência dos dados foi utilizado o Mongodb[8]. O Mongodb é um banco de dados NoSQL ou seja, um banco de dados não relacional, que utiliza a estrutura de documentos para armazenar suas informações, sendo bastante utilizados para criação de aplicações web utilizando o Nodejs descrito nos parágrafos anteriores.

[5] site: https://handlebarsjs.com/
[6] site: http://www.passportjs.org/
[7] site: https://mongoosejs.com/
[8] site: https://www.mongodb.com/

3.5.2 API Gateway

Como descrito anteriormente, o API Gateway é uma ferramenta utilizada para gerenciar as entradas de saídas de dados da aplicação funcionando como um "portão" por onde são recebidas e distribuídas as requisições vindas do ambiente externo da aplicação.

Existem diversos tipos de API Gateway no mercado atualmente, alguns deles são o Kong[9] e o Zull[10].

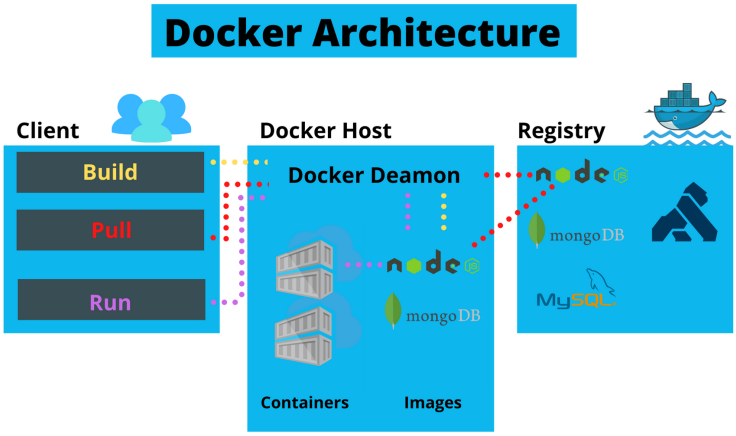
Para a criação do API Gateway utilizou-se o framework express descrito na seção 3.5.1.

3.5.3 Docker

Docker é uma plataforma capaz de criar e administrar ambientes isolados chamados de contêineres. Com ele podemos "empacotar" aplicações dentro desses ambientes e torná-las portáteis e executar em qualquer sistema que suporte o Docker. Com essa portabilidade, podemos criar, copiar, migrar e implantar qualquer aplicação com mais facilidade.

O Docker utiliza uma arquitetura cliente-servidor, que consiste em três partes, Docker Client (Figura 11), Docker Host e Registry. Em termos básicos o Docker Client comunica-se com o Docker Daemon que é um componente do Docker Host responsável por construir, executar e distribuir os containers Docker. O Docker Client pode ser executado no mesmo sistema que o Deamon estiver sendo executado, ou, ele pode conectar-se a um Daemon remoto do Docker.

Figura 11 – Arquitetura Docker



Fonte: Adaptado de
https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture/

[9] site: https://konghq.com/
[10] site: https://github.com/Netflix/zuul

O Client Docker é responsável por ser uma interface onde os usuários inserem comandos para serem executados pelo Daemon, é a principal maneira do usuário interagir com o ambiente de execução Docker. Vários comandos podem ser utilizados, como docker build, run e etc. Um dos principais recursos do Docker Client é que ele pode facilmente se comunicar com vários Daemons.

O Docker Host é responsável basicamente por executar o Docker Daemon, e assim executar solicitações vindas da API Docker, como run e build. Além disso, é responsável por gerenciar todos os containers, imagens e redes que estão dentro de seu escopo.

O Docker Registry é uma aplicação server-side responsável por armazenar e distribuir imagens dockerizadas, ou seja imagens de aplicações criadas para serem executadas em ambientes Docker. Nele podemos armazenar imagens criadas pelo usuário e obter imagens de registros públicos, funcionando como um verdadeiro repositório de imagens Docker.

3.5.4 Banco de Dados

Banco de dados são conjunto de informações relacionadas de forma a criar algum tipo de sentido, permitindo posteriormente que se faça uma consulta afim de se resgatar os dados ali armazenados. Existem diversos modelos de bancos de dados, relacional, NoSQL, hierárquico, etc. Neste trabalho, optou-se por um banco de dados orientado a documentos chamado MongoDB.

O MongoDB é um banco de dados que possui código aberto desenvolvido em 2007 pela 10gen que atualmente se chama MongoDB Inc. Sua primeira versão foi publicada em 2009. O MongoDB utiliza uma estrutura de documentos semelhantes ao JSON para armazenar seus dados. Essas características permitem que as aplicações modelem informações de modo muito mais natural.

3.5.5 Perfis de Aprendizagem - KOLB

De acordo com algumas teorias, pessoas diferentes respondem melhor a certos tipos de estímulos do que outros. Isso significa que a maneira que o conteúdo é transmitido pode fazer muita diferença em sua assimilação.

Vários estudos apontam que a maneira com que nós seres humanos aprendemos é única uns dos outros, ou seja, cada pessoa tem sua própria maneira de aprender. O que une umas pessoas das outras são certas características comuns. Nesse sentido, alguns autores buscaram propor teorias que qualificam estilos de aprendizado. Neste trabalho, usaremos uma teoria proposta por Kolb (KOLB, 1984).

A teoria proposta por Kolb qualifica os perfis de aprendizado em 4 categorias:

A primeira categoria é dos Adaptadores. Nessa categoria Kolb explica que as pessoas aprendem de maneira mais significativa através de atividades práticas, priorizando a tentativa e erro, utilizando a intuição, assumindo riscos e tomando a iniciativa.

A segunda categoria é a dos Assimiladores. Pessoas com esse perfil têm um comportamento mais reflexivo, gostam de analisar e refletir sobre o conteúdo. Estes indivíduos têm uma afinidade maior com ideias abstratas.

A terceira categoria é a dos Divergentes. Esse perfil é composto de indivíduos com uma maior capacidade imaginativa e criativa. São pessoas mais empáticas e emocionais, têm maior afinidade em trabalhar em equipe e aprendem melhor através de sensações e emoções.

A última categoria é a dos Convergentes. Indivíduos com perfil convergente têm maior facilidade em aprender com a aplicação das idéias, são bons em tomada de decisão e resolução de problemas.

4 Arquitetura Proposta

A abordagem utilizada para o desenvolvimento deste trabalho é do tipo exploratória, a fim de observar e analisar como o uso de um novo tipo de arquitetura de sistema se comporta em conjunto com outras aplicações e quais o benefícios trazidos por elas.

As ferramentas utilizadas foram escolhidas com base em seu uso pela comunidade de desenvolvimento, pelo tipo de abordagem na solução dos problemas, e pela facilidade na sua implementação e manutenção.

Como linguagem de programação padrão, optou-se pelo uso da linguagem javascript por ser uma linguagem de script de alto nível e bastante utilizada nas aplicações web em geral. Atrelado ao Javascript usou-se o Nodejs como ambiente de execução para auxiliar no desenvolvimento. A escolha dessa ferramenta veio pela facilidade em seu desenvolvimento e a quantidade de material auxiliar que ajuda na resolução de diversos problemas que poderiam ocorrer durante o seu desenvolvimento, e cujas respostas poderiam ser encontradas com mais facilidade.

Toda a arquitetura proposta apresentada na seção seguinte, com exceção da API Externa, executa em contêineres separados do Docker 3.5.3. O protocolo de comunicação utilizado para estabelecer uma comunicação entre os contêineres é o HTTP por ser bastante confiável e seguro. E para fazer as requisições entre os microserviços utilizando esse protocolo lançou-se mão do axios, que é um client HTTP baseado em promisses, ideal para requisições assíncronas, e geralmente utilizado para a construção de microserviços que utilizam esse tipo de requisição.

4.1 Componentes

A aplicação proposta neste trabalho tem sua arquitetura ilustrada na Figura 12. Nela podemos perceber que há as seguintes divisões:

1. User Aplication: Que consiste na aplicação utilizada pelo usuário. Nela consta toda a parte de template das paginas e estilos, além das chamadas dos endpoints do sistema.

2. API Gateway: Aplicação descrita na seção 3.5.2 que faz o intercâmbio entre a parte interna do sistema e a aplicação do usuário.

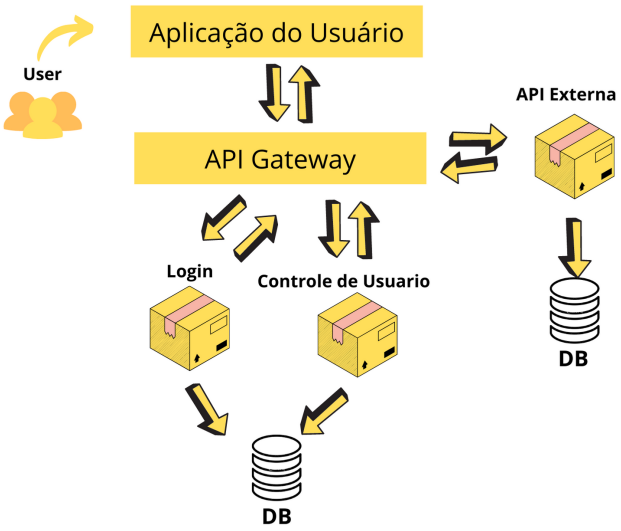
3. Serviço de Autenticação: Serviço responsável por controlar a autenticação do usuário no sistema. Este microserviço possui um banco de dados compartilhado apenas com o User Control, pela própria natureza em comum dos microserviços. Ambos lidam com os dados de acesso do usuário. Todas as rotas que passam pelo API Gateway

precisam que o usuário esteja autenticado no sistema, logo, antes de efetuar qualquer requisição interna do sistema, sua autenticação é avaliada e caso não esteja logado, a requisição não é feita, e o usuário e redirecionado para a tela de login.

4. User Control: aplicação responsável por atender as requisições referentes ao controle de usuários, ou seja, consultar usuários, deletar, registrar e editar.

5. External Aplication

Figura 12 – Arquitetura Proposta



Fonte: Elaborada pelo autor.

4.1.1 User Aplication

A aplicação do usuário é um microsserviço externo o API Gateway pois é a interface do usuário, local onde o mesmo faz as requisições, obtém, processa e se necessário renderiza as respostas na tela. Este serviço possui duas camadas, o frontend, que é a interface gráfica do sistema e o backend que é responsável por efetuar as requisições para o API Gateway.

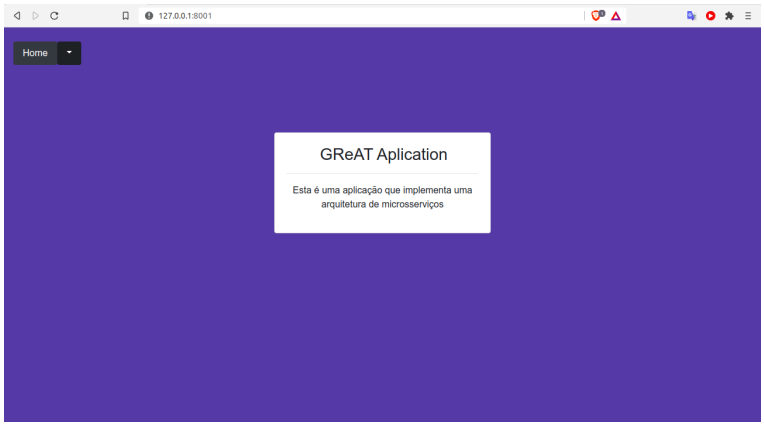
O frontend foi feito utilizando o Handlebars que é um gerenciador de templates. Nele é possível criar modelos semânticos de forma fácil e eficaz, compartimentalizando a pagina em diferentes arquivos. Isso torna possível a construção das páginas utilizando estruturas pré-prontas como menus, rodapés, e modificando apenas o conteúdo central da página, ou qualquer parte que se deseja.

O backend foi feito utilizando express que é um framework para aplicações web para Node.js, nele foi configurado as rotas e seus comportamentos, além do gerenciamento do frontend.

A página inicial contém uma breve descrição sobre a ferramenta Figura 13, um menu no canto superior esquerdo que indica quais ferramentas temos acesso. Em seu

primeiro acesso, pelo fato de o usuário não estar autenticado no sistema, ao tentar acessar a Página Inicial, o mesmo é redirecionado para a página de login onde precisa colocar suas credenciais, o mesmo acontece se o usuário tentar acessar qualquer URL do sistema não estando autenticado, isso ocorre pois todas as requisições possuem um Middleware que verifica se o usuário tem acesso a aquela url, funcionando como um protótipo do que seria um Autorization Service descrito na seção 3.3.

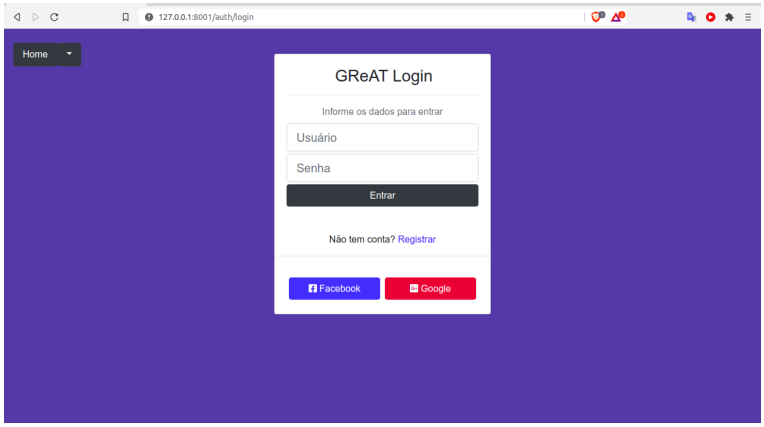
Figura 13 – Tela inicial



Fonte: Elaborada pelo autor.

Ao logar-se no sistema preenchendo suas credenciais Figura 14, o usuário é redirecionado para a página inicial descrita no parágrafo anterior. Após isso, no menu é possível identificar os botões para as páginas ao qual o usuário tem acesso, que são, caso o usuário seja um administrador: Home, Registro de Usuário, Consulta de Usuário e Logout. Caso o usuário não seja um administrador, apenas os botões de Home, Alterar Usuário, Learning Style e Logout serão visíveis. É importante destacar que, caso o usuário seja um administrador, na seção de consulta de usuários, é possível consultar qualquer usuário, e com isso deletá-lo, ou alterar suas credenciais. Isso não é possível, por medida de segurança se for um usuário comum.

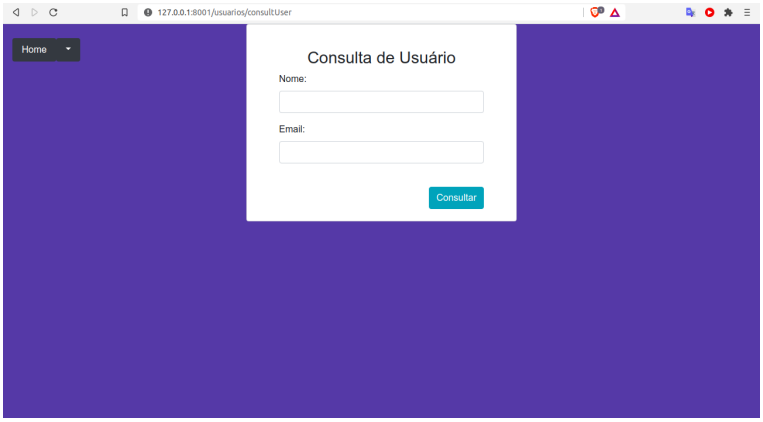
Figura 14 – Página de Login



Fonte: Elaborada pelo autor.

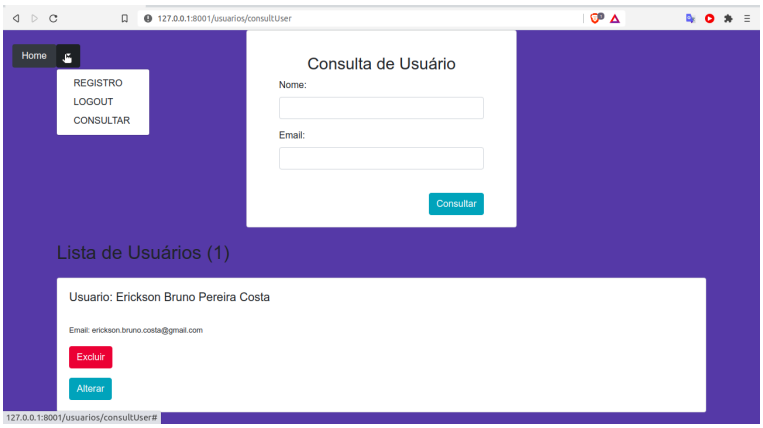
Na página de consulta de usuários Figura 15, podemos consultar utilizando dois filtros, nome e e-mail, ambos são opcionais o que indica que a consulta vazia trará todos os usuários do sistema Figura 16. Após a consulta o resultado é renderizado na página, onde é possível para da resultado, deletar o registro ou alterar os dados. Caso opte-se por alterar os dados, o usuário administrador é redirecionado para a página de alteração de dados.

Figura 15 – Página de Consulta de Usuários



Fonte: Elaborada pelo autor.

Figura 16 – Exemplo de Consulta de Usuários



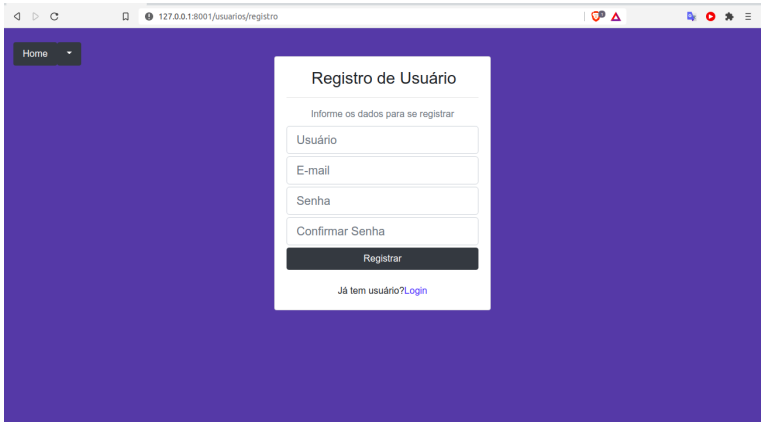
Fonte: Elaborada pelo autor.

Uma vez carregada a página de alteração de dados. Todos os campos, e-mail, nome já vem preenchidos com as informações salvas, além desses campo, tem também os campos de senha e confirmação da senha, que, por segurança não estão preenchidos. Após o preenchimento dos campos é possível salvar as alterações clicando no botão de salvar que os dados são persistidos no banco.

Caso o usuário não seja um administrador, é possível apenas alterar os dados do próprio usuário, clicando em alterar dados no menu de opções, e seguindo os passos de alteração descritos no parágrafo anterior

Para registrar um novo usuário Figura 17, o usuário logado precisa ser um administrador, preenchendo este quesito, basta apenas clicar em registrar. A tela de registro possui 4 campos que são: Nome, Email, senha e confirmação da senha. Todos os campos são de preenchimento obrigatório, uma vez preenchido basta clicar em cadastrar que assim o novo usuário será cadastrado no sistema.

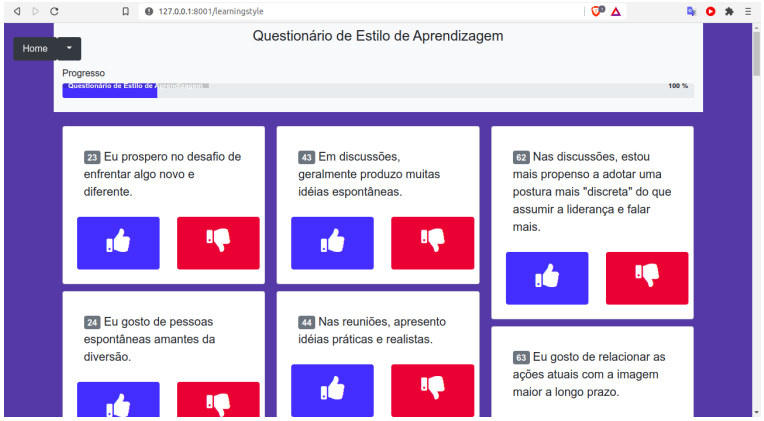
Figura 17 – Exemplo de Consulta de Usuários



Fonte: Elaborada pelo autor.

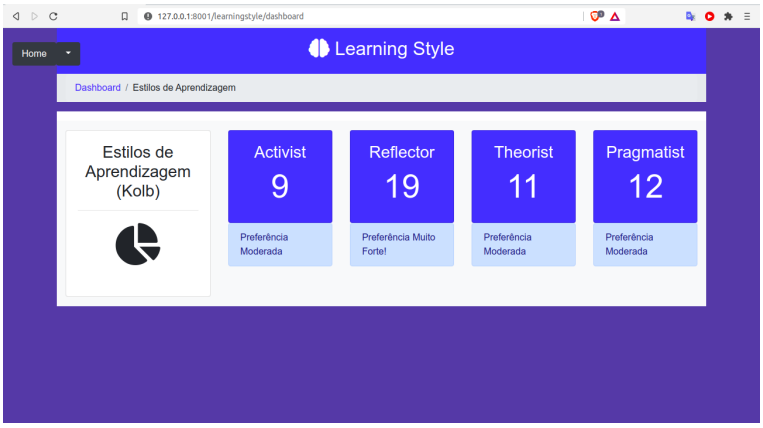
Conforme descrito na seção 3.5.5 saber qual o perfil de aprendizado dos alunos é importante para então prover a melhor forma para a qual o aluno vai estudar para absorver melhor o conteúdo. Para isso precisamos primeiramente descobrir qual é este perfil. A página de Learning Style Figura 18 inicialmente aplica um questionário proposto por (OLIVEIRA; TEIXEIRA; NETO, 2020) contendo 80 questões com respostas entre sim e não, e que aqui são representadas pelo botão de like e dislike respectivamente. Após responder todo o questionário, obtemos o perfil predominante, exibido em um dashboard na figura 19. Este questionário é aplicado uma única vez para cada usuário, de forma que depois de tê-lo feito e clicando no botão Learning Style do menu é apresentado apenas o dashboard contendo o resultado e não mais o questionário.

Figura 18 – Learning Style Questions



Fonte: Elaborada pelo autor.

Figura 19 – Learning Style Dashboard



Fonte: Elaborada pelo autor.

Tabela 1 – Requisições da Aplicação do Usuário para a API de Login

URL	**Ação**	**Atributos**	**Retorno**
POST */auth/login*	Envia os dados para autenticação.		200
GET */auth/logout*	Desloga o usuário.		200

Fonte: Elaborado pelo autor

Para deslogar do sistema basta o usuário clicar em Logout no menu, e assim o usuário não estará mais autenticado no sistema, não podendo mais acessar nenhuma aplicação interna.

4.1.1.1 User Aplication - Endpoints

A seção 4.1.1 descreve como o cada página do frontend na aplicação do usuário funciona. Esta seção se preocupa em descrever o comportamento das requisições.

Classificamos aqui os as requisições em três categorias, Serviço de Autenticação, Controle de Usuários e Api externa, cada categoria foi dividida com base no microsserviço ao qual deseja-se interagir, mesmo que de fato as requisições não sejam feitas diretamente aos microsserviços e sim o Api Gateway, processo descrito na seção 3.3.

Há duas requisições da categoria Serviço de Autenticação descritos na tabela 1, a primeira é do tipo POST onde o corpo contem o login e a senha do usuário que deseja autenticar-se no sistema, tendo como resposta o código de sucesso 201. Esta requisição é feita para o API Gateway que por sua vez verifica qual o endereço da aplicação ao qual se direciona a requisição inicial e monta-a para a aplicação de login, recebe a resposta e encaminha para a aplicação do usuário e salva o login na seção.

A outra requisição da categoria é o de logout, que é do tipo GET que remove da seção os dados do usuário logado, e redireciona para a pagina de login.

Tabela 2 – Requisições da Aplicação do Usuário para Controle de Usuário

URL	Ação	Atributos	Retorno
GET */usuarios/consultUser/{user_id}*	Envia o user_id.	{user_id}	200
POST */usuarios/consultUser/*	Envia os dados para consultar os usuários.		201
POST */user/registro*	Envia os dados para cadastrar um usuário.		201
PUT */user/edit/{user_id}*	Envia os dados para alterar um usuário.		204
DELETE */user/exc/{user_id}*	Deleta um usuário.		204

Fonte: Elaborado pelo autor

As requisições da categoria Controle de Usuário enviadas para o API Gateway são mostrados na tabela 2, começando pela requisição do tipo GET /usuarios/consultUser/{user_id} esta requisição passa por meio da url o id do usuário logado e retorna com os dados completos do mesmo, é utilizada na pagina de consulta do usuário feita por um usuário que não é um administrador, o código de retorno dessa requisição é 200 e indica sucesso.

Tabela 3 – Requisições da Aplicação do Usuário para API Externa

URL	Ação	Atributos	Retorno
GET */questions/{email}*	Consulta as questões do formulário.	{email}	200
GET */status/{email}*	Consulta os resultados do questionário.	{email}	200
POST */answers/{email}*	Envia as respostas das questões.	{email}	201

Fonte: Elaborado pelo autor

A segunda requisição da categoria Controle de usuários é do tipo POST /user/consultUser nela é passado os dados de nome e email. É utilizada na aplicação de consulta de usuários feita por um usuário administrador. Seu codigo de retorno que indica sucesso é o 201. Já a terceira requisição descrita na tabela 3, é uma requisição do tipo POST /user/registro tem em seu corpo os dados referentes a um novo usuário, esta requisição serve para um usuário administrador registrar um novo usuário na aplicação.

Outra requisição dessa categoria é do tipo PUT /user/edit, em seu corpo contém os dados alterados de um usuário, esta requisição é feita pela aplicação de alteração de usuários e tem como retorno 204 indicando sucesso na requisição.

A última requisição desta categoria é do tipo DELETE /user/exc/{user_id} e serve para um usuário administrador deletar um usuário do sistema. Tem código de sucesso 204.

A categoria Api externa são compostos pelo seguintes endpoints: O primeiro é uma requisição do tipo GET /questions/email e serve para obter as questões não respondidas pelo usuário. Ela é necessária para se saber quais perguntas o usuário já respondeu pois

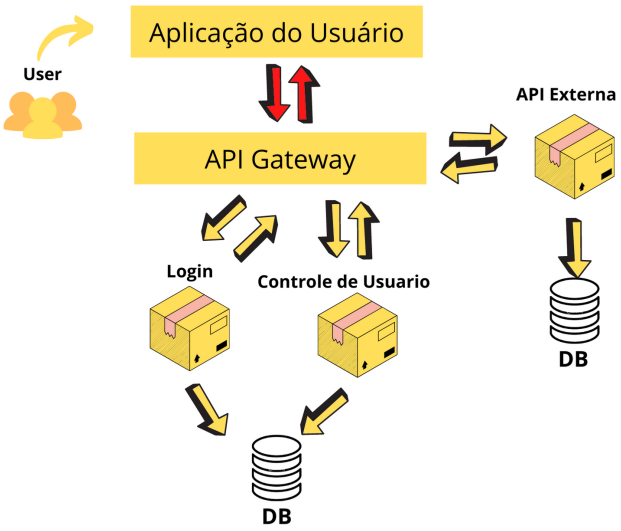
um usuário pode a qualquer momento durante a resposta sair da página e tentar responder de novo em outro momento, sendo assim a pergunta já respondida não será mais exibida.

Caso o usuário tenha respondido a todas as questões ele é redirecionado para a página de Dashboard, onde seu perfil dominante será exibido. Para obter seus dados, uma requisição do tipo GET status/email é enviada e depois de obter as respostas é renderizado na página.

O ultimo endpoint da categoria é do tipo POST /answers/email tem como parametro o email e tem em seu corpo o id da questão e a resposta dada. É responsável por enviar as respostas dadas pelo usuário para serem cadastradas pela API Externa.

Todas as requisições descritas nesta seção são direcionadas o API Gateway, que as encaminhará para seus respectivos microsserviços como mostra a Figura 21, e se necessário, dividindo em várias requisições diferentes para obter os dados das fontes corretas e montando a resposta para a aplicação do usuário.

Figura 20 – Requisições Aplicação do Usuário



Fonte: Elaborada pelo autor.

4.1.2 API Gateway

O API Gateway, descrita na seção 3.5.2 é uma aplicação utilizada na arquitetura de microserviços como um agregador de requisições vindas do ambiente externo. Nesta seção falaremos do API Gateway implementada neste trabalho.

Assim como na seção anterior, dividirei as requisições vindas da aplicação do usuário em três categorias, Serviço de Autenticação, Controle de usuário e API Externa, e são descritas a seguir tabela 4.

Tabela 4 – Endpoints do Controle de Usuários

URL	**Ação**	**Atributos**	**Retorno**
POST */auth/login*	Envia os dados para autenticação.		201
GET */answers/{user_id}*	Obtém as respostas do usuário.	user_id	200
GET */questions/*	Obtém todas as questões.		200
GET */status/{ email }*	Obtém o status do usuário.	email	200
GET */status/{user_id}*	Obtém o status do usuário.	user_id	200
POST */anwsers/{email}*	envia as respostas do usuário.	email	201

Fonte: Elaborado pelo autor

Falando das requisições vindas da aplicação do usuário, temos a primeira categoria, composta por apenas uma requisição do tipo POST /auth/login ela é responsável por requisitar uma autenticação no sistema, o API Gateway monta a requisição para o Serviço de Autenticação obtendo seu endereço válido e porta, adicionando o corpo, composto por email e senha e enviando para o microsserviço correto, obtendo a resposta e encaminhando para a aplicação do usuário.

As requisições da aplicação do usuário seguem o escopo das requisições descritas na seção anterior com a diferença que o endereço e porta do microsserviço de controle de usuário são diferentes. O API Gateway monta as requisições e as encaminha para o respectivo microsserviço, obtendo a resposta e encaminhando para a aplicação do usuário.

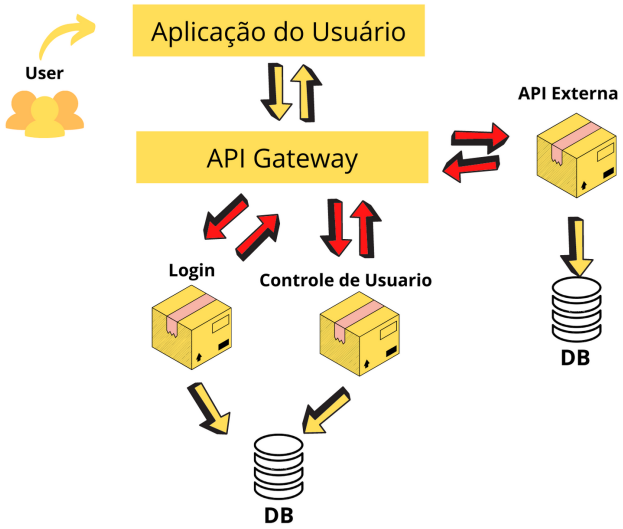
Falando agora das requisições da categoria API Externa, a requisição do tipo GET /questions/email vinda da aplicação do usuário, temos que esta requisição é na verdade uma composição de várias requisições internas. Ela é acionada pela aplicação do usuário quando o mesmo clica no botão learnig style do menu, o que quer dizer que o usuário deseja responder o questionário e assim obter seu perfil de aprendizado.

Primeiramente o API Gateway envia uma requisição do tipo GET /user/email para obter o idexterno do usuário enviando o email. Com o id do usuário em mãos, uma outra requisição do tipo GET /answers/{user_id}é enviada para obter as respostas do usuário. Com as respostas em mãos, uma ultima requisição do tipo GET questions/ é enviada para obter todas as questões, assim de posse desses dados, uma função interna do API Gateway é acionada para saber quais questões o usuário não respondeu e depois retorna elas para que seja respondidas. Caso o usuário tenha respondido todas, apenas o código de sucesso é enviado.

A segunda requisições da categoria API Externa é do tipo GET /status/email é enviada pela aplicação do usuário para saber o perfil de aprendizado predominante do usuário. Primeiramente, uma requisição do tipo GET /usuarios/email é enviada pra saber o id do usuário, logo após isso uma requisição do tipo GET /status/id é enviada pra saber o status do usuário, então a aplicação do usuário é respondida com o resultado.

A ultima requisição é do tipo POST anwser/email tem em seu corpo o id da questão e a resposta e serve para informar qual a resposta do usuário para uma determinada questão.

Figura 21 – Requisições API Gateway

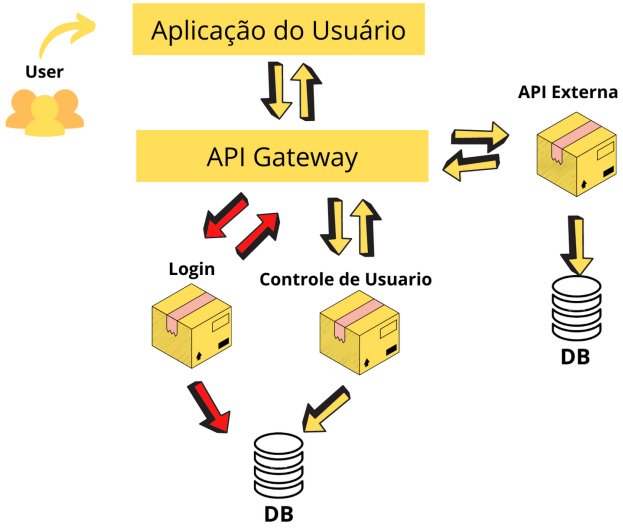


Fonte: Elaborada pelo autor.

4.1.3 Serviço de Autenticação

O microsserviço de autenticação é responsável por validar as credenciais do usuário. Comunica-se apenas com o API Gateway e seu banco de dados Figura 22.

Figura 22 – Requisições do Serviço de Autenticação



Fonte: Elaborada pelo autor.

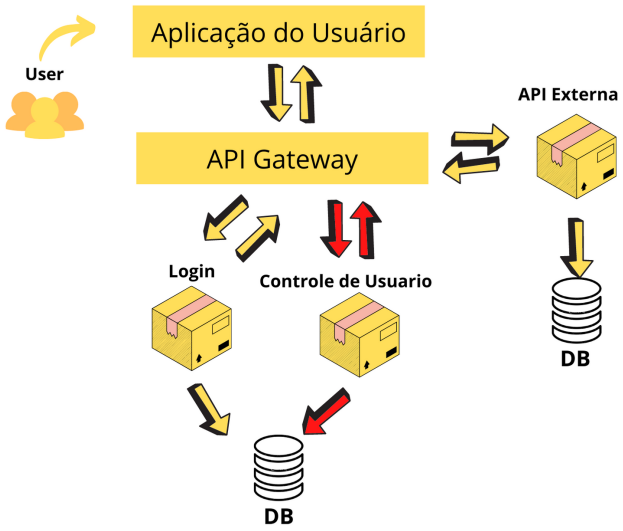
Possui apenas uma requisição do tipo POST auth/login tabela 1 e tem em seu corpo o email e senha do usuário e serve para validar se existe um usuário com as credenciais

informadas no banco de dados. A primeira coisa a ser verificada é se existe um email igual ao informado depois se o hash da senha é compatível com a salva no banco. Se ambas as comparações forem válidas então o usuário pode entrar no sistema e uma resposta de sucesso é enviada o API Gateway.

4.1.4 Controle de Usuários

O microsserviço de controle de usuários é responsável por obter, deletar, alterar e incluir as informações referentes aos usuários. Se comunica com o API Gateway e o banco de dados Figura 23, que é compartilhado com o microsserviço de login por conta da natureza dos dados armazenados, que podem ser usados para checar as credenciais, no caso do microsserviço de autenticação e a manipulação desses dados pelo microsserviço de controle de usuários. Este microsserviço possui 5 endpoints Tabela 5 que serão descritos nos parágrafos a seguir.

Figura 23 – Requisições Controle de Usuário



Fonte: Elaborada pelo autor.

O primeiro endpoint é do tipo GET / usuarios/consultUser/{user_id} e serve para obter os dados do usuário logado enviando como parametro o id do usuário. É originada ao carregar a página de consulta de usuários por um usuário que não seja um administrador. A consulta é feita no banco, com base em seu id, e os resultados obtidos são retornados para o API Gateway, assim como o código de sucesso 200.

Outro tipo de consulta de usuários é feito através de uma requisição do tipo POST enviando para o microsserviço um formulário contendo o email e o nome do usuário ao qual se deseja obter as informações. Um formulário vazio pode ser enviado para se obter um resultado contendo todos os registros de usuário. Esta requisição se origina na tela de consulta de usuários feita por um usuário administrador. O resultado da consulta é enviado para o API Gateway juntamente com o código de sucesso 200.

Para registrar um novo usuário, uma requisição do tipo POST /user/registro é enviada para o microsserviço com um formulário contendo os dados do novo usuário. Uma mensagem de sucesso é enviada com o código 201 indicando que o usuário foi incluído nos registros.

Para editar os dados de um usuário uma requisição do tipo PUT /user/edit/ contendo os dados ao qual se deseja alterar é enviada ao microsserviço, as informações são salvas no banco e depois uma mensagem de sucesso é enviada o API Gateway com o código 204.

A ultima requisição é do tipo DELETE /user/exc/{user_id} serve para deletar um usuário do banco. Apenas um usuário administrador poderá enviar uma requisição deste tipo. Ela tem como código de retorno 204 indicando sucesso.

Tabela 5 – Endpoints da Aplicação Controle de Usuários

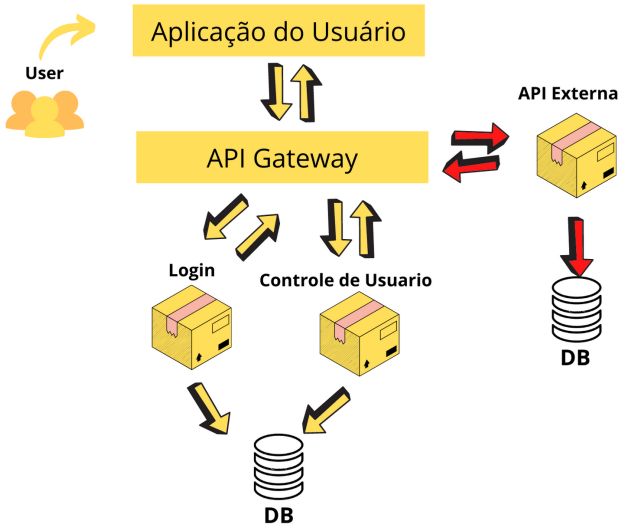
URL	**Ação**	**Atributos**	**Retorno**
GET */usuarios/consultUser/{user_id}*	Envia o user_id.	{user_id}	200
POST */usuarios/consultUser/*	Envia os dados para consultar os usuários.		201
POST */user/registro*	Envia os dados para cadastrar um usuário.		201
PUT */user/edit/{user_id}*	Envia os dados para alterar um usuário.		204
DELETE */user/exc/{user_id}*	Deleta um usuário.		204

Fonte: Elaborado pelo autor

4.1.5 API Externa

A API Externa foi desenvolvida para fazer uma ligação entre a aplicação descrita neste trabalho, e a plataforma de ensino GReAT, a fim de obter as respostas do questionário proposto por (HONEY.; MUMFORD, 2001) e assim definir qual o perfil de aprendizado do usuário. Comunica-se apenas com o API Gateway Figura 24 e todos os endpoints já foram descritos na seção 4.1.2.

Figura 24 – Fluxo Requisições API Externa



Fonte: Elaborada pelo autor.

5 Resultados

Ao buscar as informações que explicam os conceitos de arquitetura de microsserviços, no deparamos com diversos artigos que utilizam a comparação com outros tipos de arquitetura para explicar por qual motivo o uso deste novo tipo de arquitetura vem sendo bastante utilizado. E é de se imaginar o porquê, uma vez que com uma arquitetura bastante flexível como esta, as possibilidades em se criar novas tecnologias são amplificadas.

A aplicação apresentada neste trabalho foi construída com o intuito de se fazer uma prova de conceito da arquitetura de microserviços. E para tal, ela foi desenvolvida como auxiliadora na obtenção dos dados necessários para a detecção dos estilos de aprendizagem dos alunos que utilizam a plataforma de ensino GReAT. A detecção desse perfil de aprendizado, torna viável para a plataforma a utilização de um sistema de recomendação de conteúdo com base nesse perfil, portanto, descobrir qual o perfil do usuário é de extrema importância.

Ao final de seu desenvolvimento podemos avaliar que a arquitetura de microsserviços pode ser utilizada de diversas formas, inclusive como aplicação auxiliar de uma outra plataforma ou aplicação, desde que se obedeça os conceitos básicos deste tipo de arquitetura.

Na aplicação do usuário, por exemplo, podemos perceber que toda a parte de manipulação dos dados e persistência foi retirada e as responsabilidades deste tipo de processamento foram postas em diversas outras aplicações, ou seja, as responsabilidades foram divididas, isso tornou tanto o desenvolvimento quanto a manutenção desta aplicação mais facilitado, pois agora, ela lida apenas com a visualização dos dados obtidos da parte interna da arquitetura.

Outro fato que podemos depreender dos resultados finais é que embora todas as responsabilidades de processamento tenham sido retiradas da aplicação do usuário e divididas entre os microsserviços, agora se tem muito mais preocupações de onde buscar os dados. Por esse motivo, a arquitetura previu a criação de um portão de entrada, ou API Gateway, que consiste em uma aplicação responsável por conectar-se aos microsserviço e assim poder buscar as informações corretamente.

Os microsserviços são, na visão deste autor, a parte mais importante deste tipo de arquitetura, pois todo o processamento bruto das informações é feito neles. Por isso é tão importante que cada microsserviço lide apenas com um tipo de processamento de dados. Isso a medida que a quantidade de microsserviços aumenta é crucial para que haja uma melhor qualidade na manutenção, além de facilitá-la.

6 Conclusão

A arquitetura apresentada na seção 3.3 foi desenvolvida como demonstrado na seção 4, utilizando-se dos conceitos da arquitetura de microsserviços. Como resultados temos que uma aplicação mais versátil e flexível, admitindo mudanças de forma mais rápida, melhor que isso, uma aplicação que pode ser integrada a uma outra plataforma que utiliza outro tipo de arquitetura sem perder as características que a tornam tão diferentes.

Como podemos demonstrar neste trabalho este novo conceito de arquitetura tem um potencial muito grande diante das diversas possibilidades as quais podemos submeta-la. E, embora seu desenvolvimento seja ainda um desafio, as qualidades apresentadas neste aqui tornam seu desenvolvimento bastante promissor.

Como trabalhos futuros, espera-se aumentar a quantidade de microsserviços internos da aplicação, agregando mais responsabilidades e fortalecendo a arquitetura. Espera-se também agregar um orquestrador de contêiner, aplicação responsável por gerenciar os microsserviços que estão sendo executados em contêineres do Docker, essa aplicação verifica a saúde dos microsserviços, executando os que ficaram inativos, escalonando os mais requisitados se necessário ou até reduzindo as suas instâncias. Tudo isso de forma automática, sem a necessidade de um desenvolvedor monitorar manualmente.

Referências

DOGLIO, F.; DOGLIO; CORRIGAN. *REST API Development with Node. js*. [S.l.]: Springer, 2018. Citado na página 18.

FIELDING, R. T.; TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine Irvine, 2000. v. 7. Citado na página 18.

FOWLER, L. *James Lewis and Martin Fowler 2014 - Microservices*. 2014. <https://www.martinfowler.com/microservices/>. Acessado em 12/10/2020. Citado na página 13.

HONEY., P.; MUMFORD, A. Honey and mumford learning styles questionnaire. *Peter Honey Learning. Retrieved April*, v. 25, p. 2007, 2001. Citado 2 vezes nas páginas 16 e 42.

INFORMAÇÃO, N. de; PONTO, B. Coordenação do. Pesquisa sobre o uso das tecnologias de informação e comunicação nos domicílios brasileiros: Tic domicílios 2019. *São Paulo: Comitê Gestor da Internet no Brasil*, 2020. Citado na página 13.

KOLB, D. *Experiential Learning: Experience As The Source Of Learning And Development*. [S.l.: s.n.], 1984. v. 1. ISBN 0132952610. Citado 2 vezes nas páginas 16 e 29.

MEIRELLES, F. *Pesquisa Anual Uso de TI nas Empresas–FGVcia–Centro de Tecnologia de Informação Aplicada da FGV-EAESP, 31ª edição, 2020*. [S.l.], 2020. Citado na página 13.

NADAREISHVILI, I.; MITRA, R.; MCLARTY, M.; AMUNDSEN, M. *Microservice architecture: aligning principles, practices, and culture*. [S.l.]: "O'Reilly Media, Inc.", 2016. Citado na página 21.

NEWMAN, S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. [S.l.]: O'Reilly Media, 2019. Citado na página 13.

OLIVEIRA, A.; NETO, C. d. S. S.; TEIXEIRA, M. M. Um ambiente de autoria de jogos sérios pelo usuário final aplicados a educação. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2014. v. 25, n. 1, p. 1058. Citado na página 16.

OLIVEIRA, A.; TEIXEIRA, M. M.; NETO, C. d. S. S. Recommendation of educational content to improve student performance: An approach based on learning styles. In: INSTICC. *CSEDU (2)*. [S.l.]: SciTePress, 2020. p. 359–365. ISBN 978-989-758-417-6. Citado 2 vezes nas páginas 16 e 35.

RICE, L. *Container Security: Fundamental Technology Concepts that Protect Containerized Applications*. [S.l.]: O'Reilly Media, 2020. Citado na página 26.

VILLAMIZAR, M.; GARCES, O.; OCHOA, L.; CASTRO, H.; SALAMANCA, L.; VERANO, M.; CASALLAS, R.; GIL, S.; VALENCIA, C.; ZAMBRANO, A. et al. Infrastructure cost comparison of running web applications in the cloud using aws

lambda and monolithic and microservice architectures. In: IEEE. *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. [S.l.], 2016. p. 179–182. Citado na página 15.