TECH CHALLENGE – FIAP

Projeto: API para serviço de Gerenciamento de Pedidos;
Curso: Arquitetura e Desenvolvimento em JAVA – Fase 4;
Escola: FIAP;

Grupo:
Victor Luiz Montibeller;
Leonardo Arantes Di Nizo;
Alexandre Marques;

Ferramentas utilizadas:
Event Storming: Miro;
Linguagem: JAVA 17;
Arquitetura: MVC, Micro Serviços;
Framework: Spring Boot, Spring JPA, Spring Batch, Spring Cloud Stream, Hibernate, Lombok;
Repositório: GitHub;
Banco de dados: H2 DataBase;
Publicação: Docker, Render;
Testes: JUnit, Mockito;
Dependências: Maven;
Documentação: Swagger;

Sumário

Introdução

O Sistema de Gerenciamento de Pedidos é uma aplicação web desenvolvida em Java, projetada para atender às necessidades dos clientes e proprietários de lojas virtuais em um mundo cada vez mais digitalizado. Com o aumento do uso de dispositivos móveis e da internet, os consumidores esperam ter acesso rápido e fácil para fazer pedidos online, monitorar o status de suas entregas e receber seus pedidos no conforto de suas casas. Além disso, os proprietários de lojas podem gerenciar eficientemente o estoque de seus produtos, otimizar o processo de preparação e entrega de pedidos, e oferecer um serviço de qualidade superior aos seus clientes.

Contexto do Projeto

Nos dias de hoje, muitos consumidores confiam em plataformas online para fazer pedidos, verificar o status das entregas e avaliar a qualidade do serviço recebido. Isso inclui a leitura de comentários de outros clientes, visualização de fotos dos produtos, e acompanhamento em tempo real do status do pedido e da entrega. Por outro lado, os proprietários das lojas buscam maneiras de atrair mais clientes, otimizar a gestão do estoque e melhorar a eficiência do serviço de entregas, garantindo que os clientes recebam seus pedidos de forma rápida e precisa.

Importância da API

A API do Sistema de Gerenciamento de Pedidos visa preencher a lacuna no gerenciamento eficiente de pedidos, controle de estoque e serviço de entregas, proporcionando uma plataforma centralizada onde clientes e estabelecimentos podem interagir de forma eficiente e conveniente. Ao oferecer recursos como pedidos online, controle de estoque em tempo real e gerenciamento de entregas, a API ajuda a melhorar a experiência do usuário e a promover a transparência e a confiança entre os consumidores e os estabelecimentos.

Objetivos do Projeto

Os principais objetivos do projeto incluem:

- Facilitar a realização de pedidos online, oferecendo aos clientes uma maneira rápida e conveniente de selecionar e comprar produtos.
- Permitir que os clientes avaliem sua experiência de compra e entrega, compartilhando feedbacks úteis para outros consumidores.
- Auxiliar os estabelecimentos a gerenciar suas operações de forma mais eficiente, permitindo o acompanhamento dos pedidos, a análise das avaliações dos clientes e o controle do estoque em tempo real.

Visão Geral do Projeto

O Sistema de Gerenciamento de Pedidos é uma aplicação web desenvolvida em Java, utilizando o framework Spring Boot. A API foi projetada para fornecer funcionalidades abrangentes para gerenciar pedidos online, controlar o estoque em tempo real e otimizar o serviço de entregas. A plataforma permite aos estabelecimentos monitorar a disponibilidade de produtos, gerenciar o fluxo de pedidos e garantir uma entrega eficiente e precisa aos clientes.

Funcionalidades Principais

A API oferece os seguintes recursos principais:

- Cadastro de Clientes: Permite que os clientes se cadastrem na plataforma fornecendo informações como nome, e-mail, e data de cadastro.
- Cadastro de Endereços: Permite o cadastro de endereços dos clientes e dos estabelecimentos, incluindo informações como rua, número, bairro, cidade, estado, país e CEP.
- Cadastro de Produtos: Permite que os estabelecimentos registrem produtos na plataforma, fornecendo detalhes como nome do produto, descrição, preço e quantidade disponível em estoque.
- Importação de Produtos: Permite importar os produtos no sistema através de arquivo .CSV ou configurar para que a rotina importe os registros automaticamente.
- Gestão de Pedidos: Os clientes podem fazer pedidos online, especificando os produtos desejados, quantidade e preferências. Os estabelecimentos podem visualizar, gerenciar e confirmar os pedidos recebidos.
- Controle de Estoque: Monitora o estoque de produtos em tempo real, alertando os estabelecimentos quando itens estão próximos de acabar e facilitando a reposição.
- Serviço de Entregas: Gerencia o processo de entrega, desde a preparação dos pedidos até a entrega ao cliente, com acompanhamento em tempo real e notificações automáticas.

Tecnologias Utilizadas

A API foi desenvolvida utilizando as seguintes tecnologias e ferramentas:

- Java: Linguagem de programação principal para o desenvolvimento da aplicação.
- Spring Boot: Framework Java usado para criar aplicativos baseados em Spring com facilidade.
- Spring Data JPA: Implementação da camada de acesso a dados baseada em JPA.
- Spring MVC: Padrão de projeto usado para desenvolver aplicativos web em Java.
- Spring Batch: Framework para execução de processamento de volumes de dados robusto, a partir de alguma fonte.
- Spring Cloud Stream: Estrutura para construir microsserviços orientados a eventos altamente escaláveis conectados a sistemas de mensagens compartilhadas.
- Hibernate: Biblioteca ORM para mapeamento objeto-relacional e persistência de dados.
- Docker: Conjunto de produtos de plataforma como serviço (PaaS) que usam virtualização de nível de sistema operacional para entregar software em pacotes chamados contêineres.
- Lombok: Framework para Java que permite escrever código eliminando a verbosidade, o que permite ganhar tempo de desenvolvimento para o que realmente é importante.
- H2: Sistema de gerenciamento de banco de dados relacional (RDBMS) escrito em Java.
- Maven: Ferramenta desenvolvida pela Apache, ela serve para gerenciar as dependências e automatizar seus builds.
- JUnit e Mockito: Frameworks de teste unitário e mocking para garantir a qualidade do software.
- Swagger: Ferramenta para projetar, construir, documentar e consumir serviços da Web RESTful.
- GitHub: Plataforma de desenvolvimento que permite aos desenvolvedores criar, armazenar, gerenciar e compartilhar seu código.

Integração com Outros Serviços

A API pode ser facilmente integrada com uma variedade de serviços externos para ampliar suas funcionalidades e proporcionar uma experiência do usuário ainda mais completa. Essas integrações podem incluir:

- Sistemas de Pagamento: Integre com plataformas de pagamento como PayPal, Stripe, PagSeguro, entre outros, para permitir transações seguras e convenientes diretamente através da sua aplicação.
- Serviços de Geolocalização: Utilize serviços como Google Maps, Mapbox ou serviços de geocodificação para oferecer recursos de localização, como busca de endereços, rotas, e visualização de mapas interativos.
- Sistemas de Notificação: Integre com serviços de notificação push (como Firebase Cloud Messaging, Apple Push Notification Service) ou serviços de e-mail (como SendGrid, Amazon SES) para enviar alertas, atualizações e comunicações personalizadas aos usuários.
- Plataformas de Análise e Monitoramento: Integre com ferramentas de análise como Google Analytics, AWS CloudWatch, ou soluções de monitoramento de desempenho para obter insights detalhados sobre o uso da sua API e a saúde da aplicação.
- Serviços de Autenticação e Autorização: Implemente integrações com provedores de identidade como OAuth (Google, Facebook, etc.) ou sistemas corporativos de autenticação para reforçar a segurança e gerenciar acessos de usuários de forma eficiente.
- Integrações Personalizadas: Desenvolva integrações personalizadas conforme as necessidades específicas da sua aplicação, conectando-se a APIs de terceiros que oferecem funcionalidades específicas desejadas pelos seus usuários.

Essas integrações não apenas estendem as funcionalidades da sua API, mas também melhoram a experiência do usuário ao proporcionar recursos adicionais, automatizar processos e manter a aplicação atualizada com as melhores práticas e tecnologias disponíveis.

Arquitetura do Software

A arquitetura do software é uma parte fundamental do projeto, pois define a estrutura geral da aplicação e como suas diferentes partes se relacionam entre si. No caso do Sistema de Gerenciamento de Pedidos, utilizamos uma combinação dos padrões MVC, Micro Serviços e Clean Architecture para garantir uma arquitetura sólida e escalável.

Padrão MVC (Model-View-Controller)

O padrão MVC é amplamente utilizado em aplicações web para separar as preocupações de apresentação, lógica de negócio e acesso a dados. Nossa implementação do MVC no projeto é a seguinte:

- Model (Modelo): Representa a camada de negócios da aplicação, incluindo entidades como Pedido, Cliente, Produto, etc. Estas entidades encapsulam os dados da aplicação e a lógica de negócio associada a eles.
- View (Visão): Responsável pela apresentação dos dados aos usuários. No contexto da API, a View é representada pelos endpoints RESTful que expõem os recursos da aplicação aos clientes.
- Controller (Controlador): Atua como intermediário entre a View e o Modelo, recebendo requisições dos clientes, invocando a lógica de negócio apropriada e retornando as respostas correspondentes. Os controllers roteiam as requisições para as ações corretas e garantem uma interação suave entre a camada de apresentação e a camada de negócio.

Clean Architecture com Microsserviços

A arquitetura do software é baseada nos princípios da Clean Architecture, com a adição da estrutura de Microsserviços, que proporciona as seguintes camadas:

- Microsserviços: Cada microsserviço encapsula funcionalidades específicas da aplicação de Gerenciamento de Pedidos e é implantado de forma independente, o que facilita a escalabilidade e a evolução contínua do sistema.
- Entities (Entidades): Representam os conceitos fundamentais do negócio de Gerenciamento de Pedidos e são compartilhadas entre os microsserviços quando necessário. As entidades encapsulam os dados e o comportamento relacionados a esses conceitos e são independentes de qualquer microsserviço específico.
- Controllers: Responsáveis por receber e rotear as requisições HTTP para os microsserviços apropriados, garantindo a interação correta entre os clientes e os serviços.

- Services (Serviços): Implementam os casos de uso específicos de cada microsserviço de Gerenciamento de Pedidos, coordenando a lógica de negócio e a interação com as entidades compartilhadas.
- Repositories (Repositórios): Responsáveis pelo acesso e persistência dos dados específicos de cada microsserviço. Eles encapsulam as operações de consulta e persistência e garantem que cada serviço tenha seu próprio armazenamento de dados isolado.

Benefícios da Arquitetura

A arquitetura baseada nos princípios da Clean Architecture, combinada com a estrutura de Microsserviços, oferece os seguintes benefícios para o sistema de Gerenciamento de Pedidos:

- Separação de Preocupações: As responsabilidades de apresentação, lógica de negócio e persistência são claramente separadas entre os microsserviços, facilitando a manutenção e evolução de cada parte do sistema de forma independente.
- Escalabilidade e Desempenho: Os microsserviços permitem escalar partes específicas da aplicação de Gerenciamento de Pedidos de acordo com a demanda, melhorando o desempenho global e a resposta a picos de tráfego.
- Resiliência e Tolerância a Falhas: Cada microsserviço pode ser projetado para ser resiliente, lidando com falhas de forma isolada e garantindo que o restante do sistema continue operacional.
- Flexibilidade Tecnológica: Cada microsserviço pode escolher a tecnologia mais adequada para sua função específica de Gerenciamento de Pedidos, adaptando-se às necessidades de negócio sem comprometer outros componentes do sistema.
- Melhoria na Implantação e Atualização: A implantação e atualização de microsserviços podem ser feitas de forma independente, reduzindo o impacto de mudanças e permitindo entregas contínuas de novas funcionalidades.
- Independência de Tecnologia: As camadas internas da aplicação são independentes de frameworks externos e detalhes de implementação, permitindo uma maior flexibilidade e adaptabilidade à medida que os requisitos do sistema mudam.
- Testabilidade: A arquitetura limpa facilita a escrita de testes automatizados, pois as diferentes partes do sistema podem ser testadas de forma isolada e sem depender de infraestrutura externa.

Em resumo, a combinação da Clean Architecture com a estrutura de Microsserviços proporciona uma arquitetura flexível, escalável e de fácil manutenção para o sistema de Gerenciamento de Pedidos, adaptada para aplicações modernas distribuídas e baseadas em nuvem.

Definição dos MicroServiços

MicroServiço de Cliente

Descrição: Responsável pela gestão dos dados dos clientes da plataforma:

- Cadastro de novos clientes.
- Atualização de informações de clientes.
- Autenticação e autorização de clientes.

MicroServiço de Produto

Descrição: Gerencia informações relacionadas aos produtos oferecidos pela plataforma:

- Cadastro e atualização de produtos.
- Catalogação e categorização de produtos.
- Gestão de estoque e disponibilidade.
- Importação de produtos através de arquivo .CSV.

MicroServiço de Pedidos

Descrição: Responsável pela gestão dos pedidos realizados pelos clientes:

- Registro de novos pedidos.
- Acompanhamento do status dos pedidos.
- Gestão de pagamentos e faturas.
- Notificações de atualizações de status.
- Histórico de pedidos dos clientes.

MicroServiço de Entregas

Descrição: Gerencia o processo de entrega dos produtos aos clientes:

- Rastreamento de pedidos em tempo real.
- Gestão de rotas e logística de entregas.
- Integração com serviços de transporte.
- Notificações de entrega aos clientes.

Cada um desses MicroServiços opera de forma independente, mas se integra com os outros para fornecer uma experiência completa ao usuário final na plataforma.

Qualidade de Software

A qualidade de software desempenha um papel fundamental na confiabilidade, segurança e eficácia da nossa API de Gerenciamento de Pedidos. Para garantir altos padrões de qualidade em nosso projeto, adotamos diversas práticas e ferramentas que promovem a detecção precoce de defeitos e a melhoria contínua do código.

Testes Unitários com JUnit e Mockito

Os testes unitários são essenciais para garantir que cada componente da nossa API funcione conforme o esperado. Utilizamos o framework JUnit para escrever e executar testes unitários de forma automatizada. Isso nos permite verificar o comportamento de métodos e classes individuais, garantindo que cada unidade de código atenda aos requisitos especificados.

Além disso, empregamos o Mockito para criar mocks de objetos durante os testes unitários. Essa prática nos permite isolar as unidades de código em teste e simular o comportamento de dependências externas, aumentando a eficiência e a confiabilidade dos testes.

Testes de Integração

Os testes de integração são cruciais para validar a interação entre os diferentes componentes da nossa API. Implementamos testes de integração para garantir que os diversos módulos da aplicação se integrem corretamente e produzam o resultado esperado. Esses testes são executados em um ambiente que replica o ambiente de produção, garantindo uma validação realista das interações entre os componentes.

Inspeção de Código

A inspeção de código é uma prática indispensável para identificar e corrigir potenciais problemas de qualidade no código fonte. Realizamos revisões de código periódicas para identificar e corrigir problemas de estilo, complexidade excessiva, bugs e outras questões de qualidade. Essa atividade colaborativa envolve membros da equipe de desenvolvimento que revisam o código uns dos outros em busca de possíveis melhorias e correções.

Cobertura de Testes com Coverage

A cobertura de testes é uma métrica importante que indica a porcentagem do código fonte que é exercida por testes automatizados. Utilizamos ferramentas de análise de cobertura, como JaCoCo, para avaliar a cobertura de testes da nossa API. Essas ferramentas nos fornecem insights sobre quais partes do código não estão sendo testadas adequadamente, ajudando a identificar lacunas na cobertura de testes que precisam ser abordadas.

Para executar os testes com cobertura, execute os comandos abaixo:

- **mvn clean test;**
- **mvn jacoco:report;**

Dentro do diretório target/site/jacoco/, você encontrará os relatórios HTML detalhando a cobertura de cada classe em seu projeto. Esses relatórios mostram a porcentagem de linhas de código cobertas por testes.

Event Storming

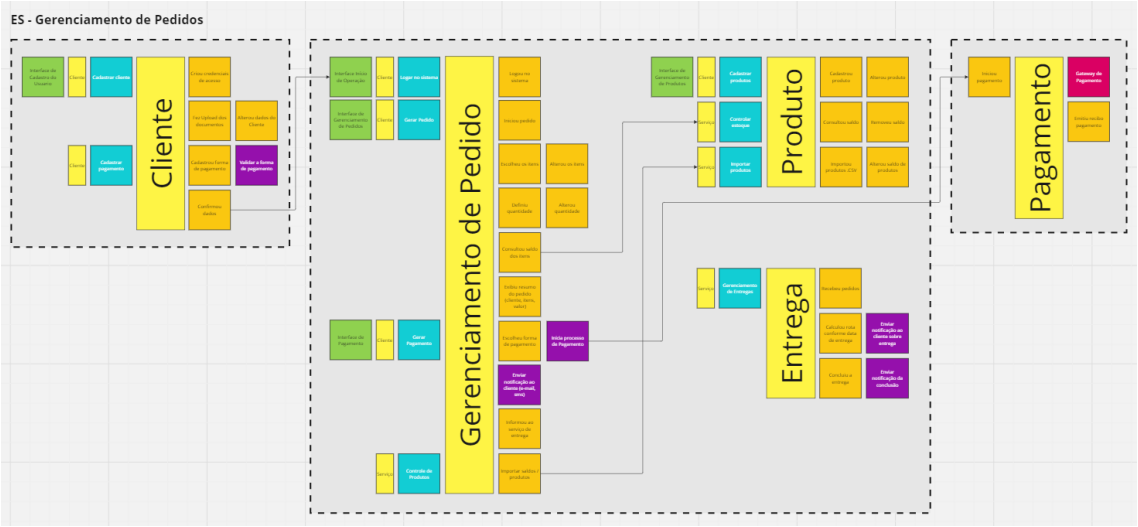
Iniciando com um brainstorming, discutindo as ideias do projeto entre os participantes do grupo que descrevessem os eventos e gatilhos da nossa API, simulando uma reunião com os especialistas e solicitantes do negócio.

Organizamos os comandos, atores, interfaces, políticas do negócio e eventos pivotais, identificando onde inicia e onde termina cada parte do processo.

Em seguida aplicamos o objetivo principal, agregados e contextos, separando-os para melhor visualização.

Em cada fase, novas ideias foram surgindo e assim fomos refinando o processo ao longo de sua construção. Todo esse processo de desenvolvimento do Event Storming e suas fases, podem ser visualizados através do link abaixo:

https://miro.com/app/board/uXjVNqrQfdY=/



Melhorias Futuras

Após a entrega do projeto funcionando com os requisitos solicitados, deixamos o planejamento para algumas melhorias futuras, como:

- Implementação de Autenticação e Autorização: Adicionar autenticação e autorização para proteger endpoints sensíveis da API, garantindo que apenas usuários autorizados possam acessá-los.
- Melhorias na Segurança: Implementar práticas de segurança adicionais, como proteção contra-ataques de injeção SQL, XSS e CSRF, além de configurar HTTPS para comunicação segura.
- Melhorias de Desempenho: Realizar otimizações de consulta e indexação no banco de dados para melhorar o desempenho da API, especialmente em cenários de alta carga.
- Implementação de Cache: Utilizar técnicas de cache para armazenar em cache dados frequentemente acessados e reduzir a carga no banco de dados, melhorando a escalabilidade e o desempenho geral da aplicação.
- Aprimoramento da Documentação da API: Refinar e expandir a documentação da API para torná-la mais abrangente e amigável para os desenvolvedores, incluindo exemplos de solicitações e respostas.
- Implementação de Testes de Integração Completa: Desenvolver testes de integração completos que cobrem todos os casos de uso principais da aplicação, garantindo uma cobertura de teste abrangente. problemas em tempo real e tomar medidas corretivas proativas.
- Adição de Recursos Adicionais: Considerar a adição de novos recursos, como suporte a múltiplos idiomas, integração com serviços de pagamento online para reservas pagas, ou funcionalidades de recomendação personalizada de restaurantes.
- Interface gráfica: Front-End para integração com a API, possibilitando uma melhor adaptação aos processos, como um app WEB e versão Móbile.
- Notificações: Enviar lembretes de notificações por SMS e e-mail, com a possibilidade de cadastrar o evento automaticamente na agenda dos usuários.

Acesso ao Projeto

O projeto da API para serviço de Gerenciamento de Pedidos, codificado seguindo o que foi definido no Objetivo do Projeto, está disponível no repositório do GitHub:

https://github.com/LeonardoArantes/TechChallenge_IV_GerenciamentoPedidos

Configurando a API

Clonar o Repositório do GitHub:

Para configurar a API, basta clonar o projeto:
https://github.com/LeonardoArantes/TechChallenge_IV_GerenciamentoPedidos

Criar e Executar uma Imagem Docker:

- Para executar o Docker Compose, você precisa usar o comando docker-compose up. Este comando irá construir, (re)criar, iniciar e anexar aos contêineres para um serviço.
    - **docker-compose up**

Utilizar uma imagem Docker pública, no Registry do Docker:

Acessar a imagem através da URL abaixo:

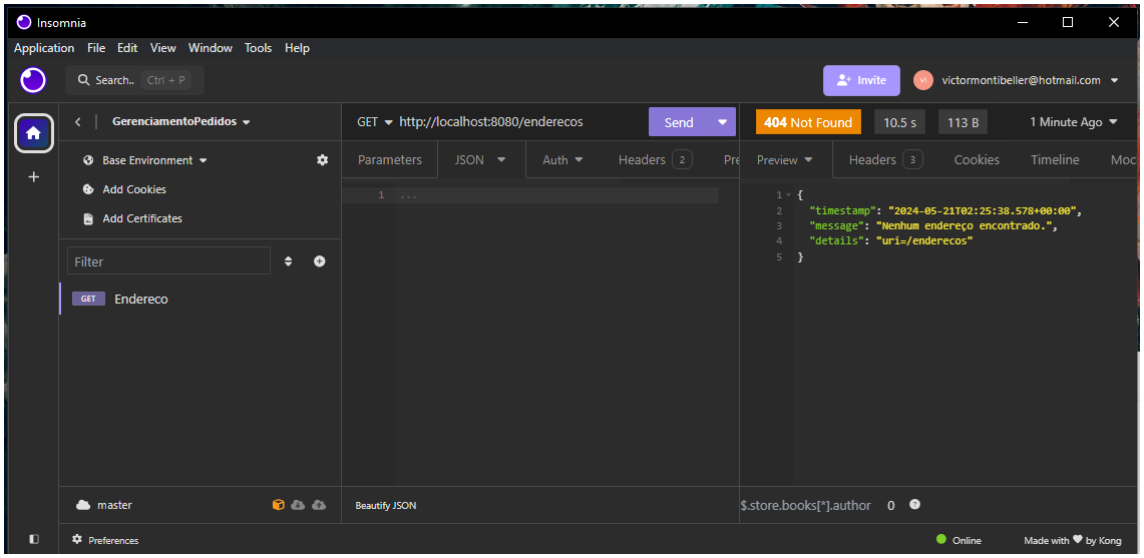
- Docker.io/victormontibeller/image_gerenciamentopedidos:latest;

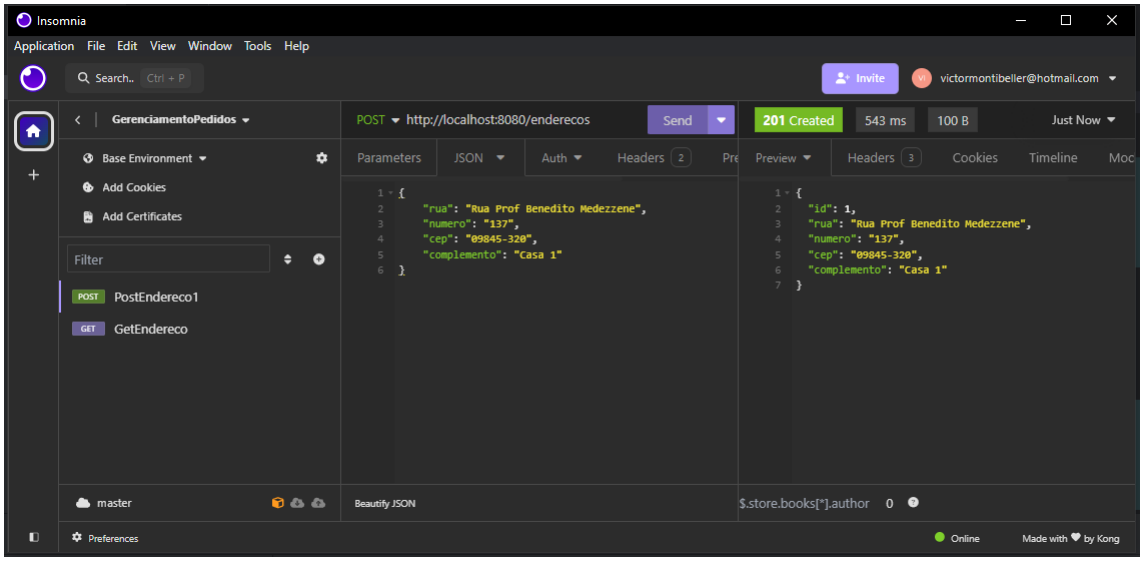
Testando a API

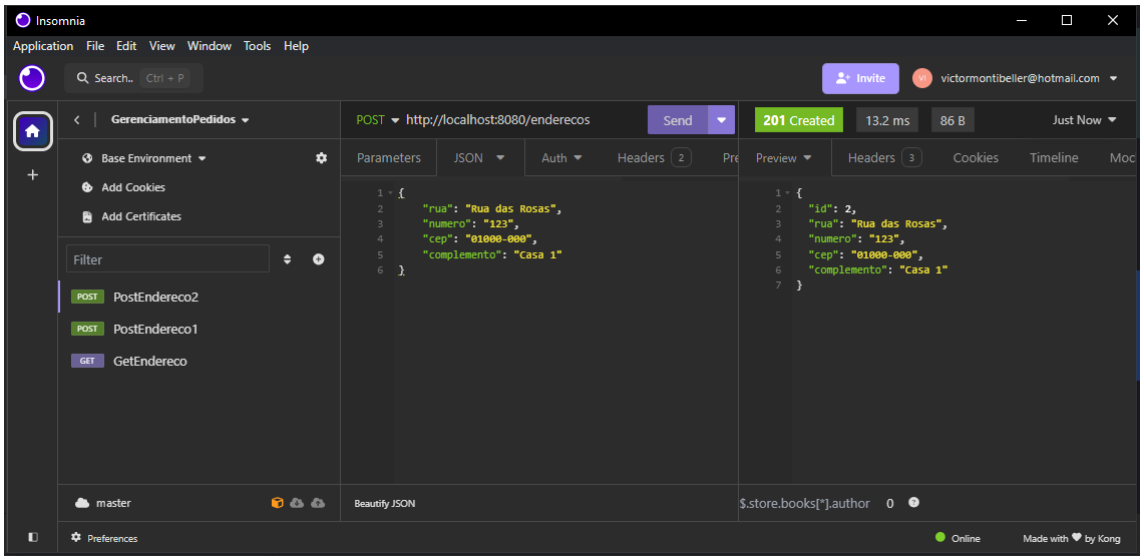
Para testar nossa API, temos alguns exemplos de payloads do tipo Json para o envio de requisições via Insomnia ou outra ferramenta semelhante. Esses payloads se encontram no projeto.

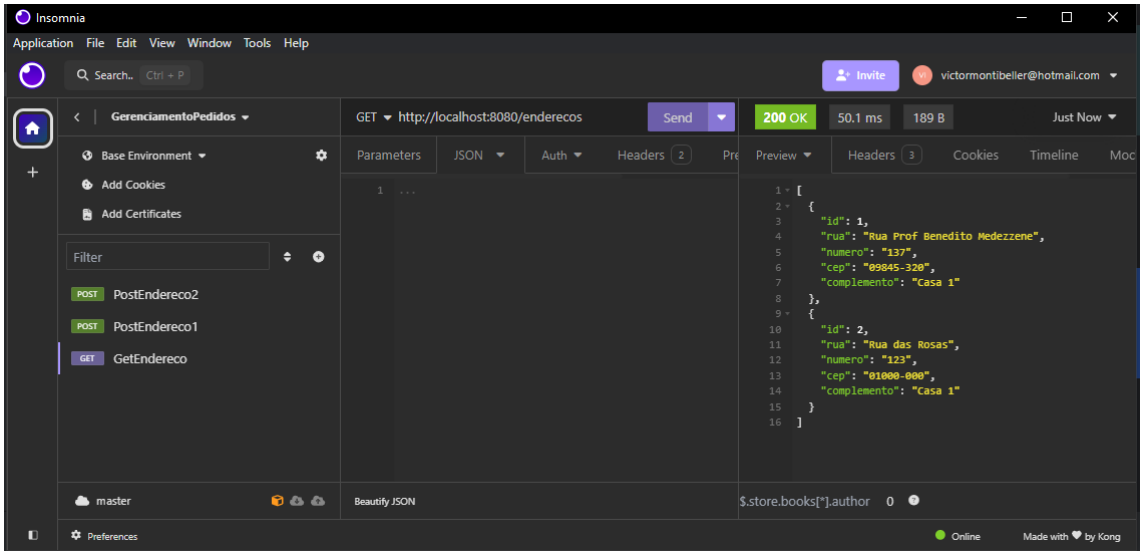
Necessário seguir o fluxo para que a aplicação tenha os dados necessários para gerar corretamente as Reservas.

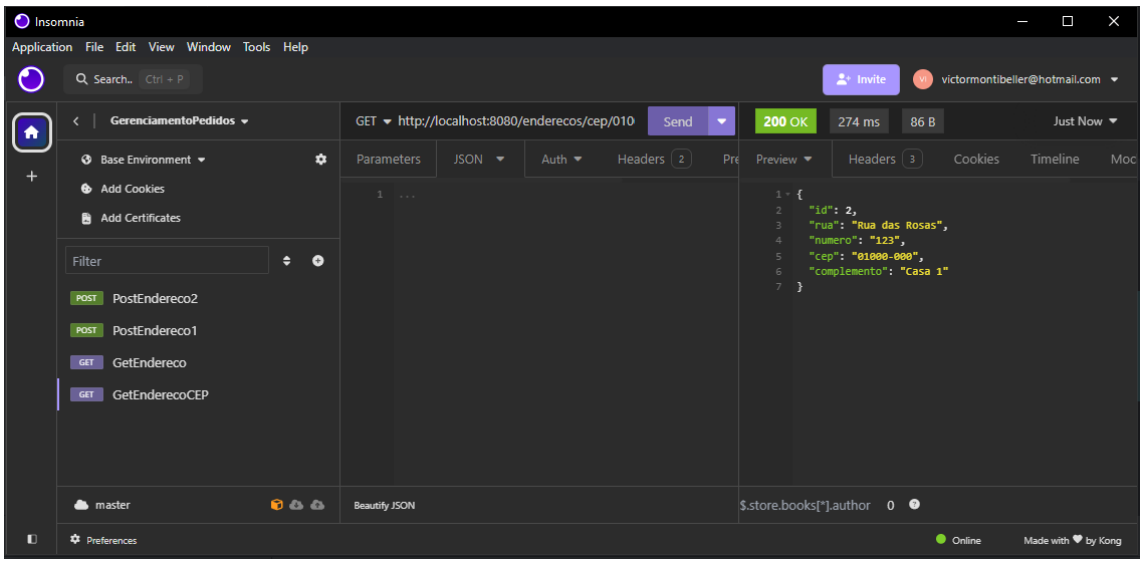
Endereços:



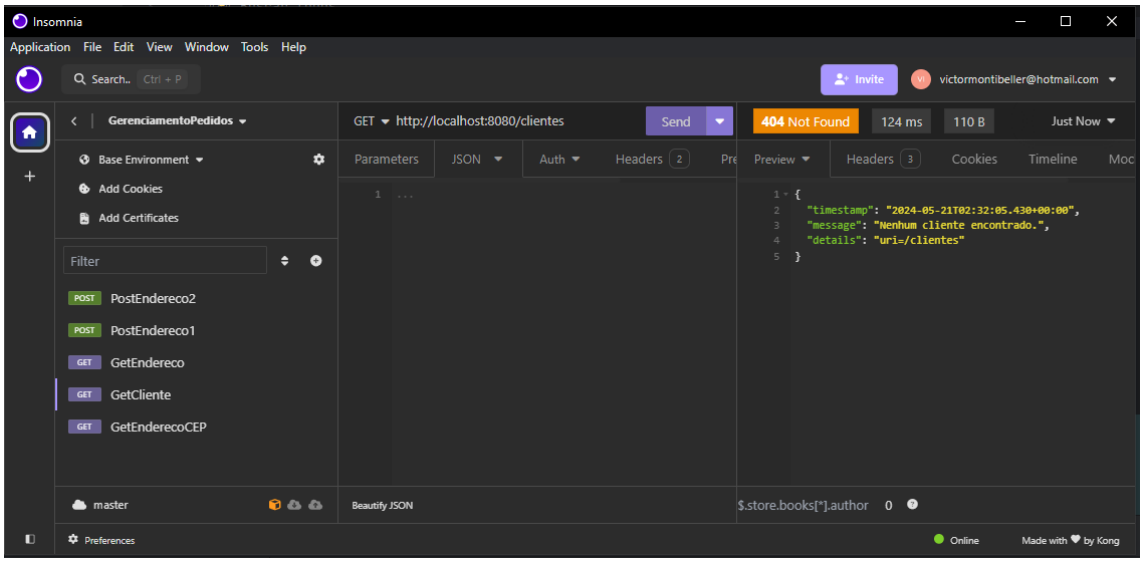


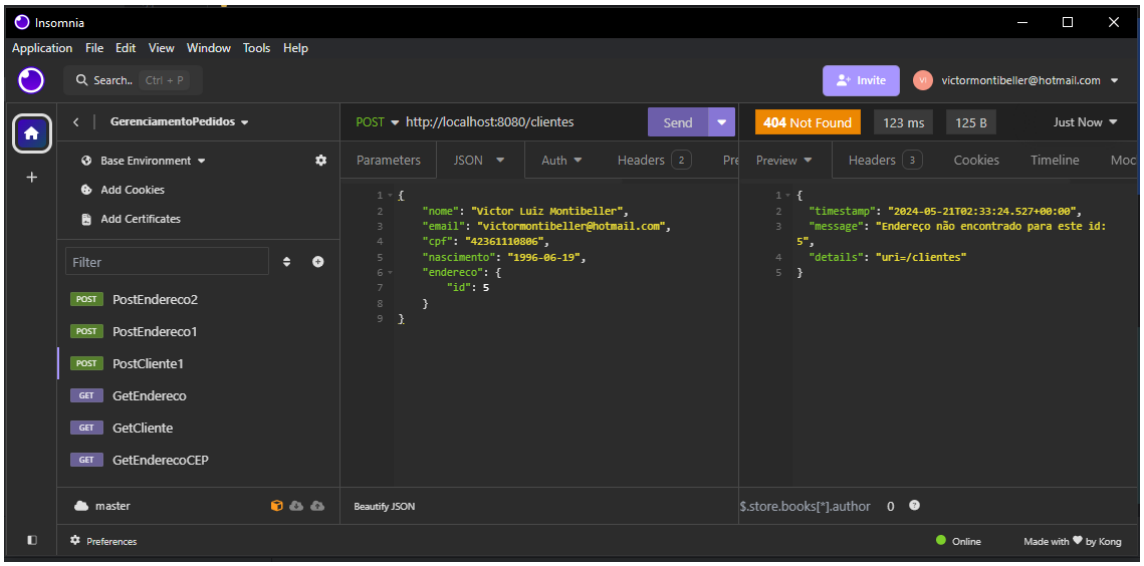

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
Base Environment
Add Cookies
Add Certificates
Filter
POST PostEndereco2
POST PostEndereco1
GET GetEndereco
master
POST http://localhost:8080/enderecos
Send
201 Created
13.2 ms
86 B
Just Now
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"rua": "Rua das Rosas",
"numero": "123",
"cep": "01000-000",
"complemento": "Casa 1"
}
{
"id": 2,
"rua": "Rua das Rosas",
"numero": "123",
"cep": "01000-000",
"complemento": "Casa 1"
}
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong

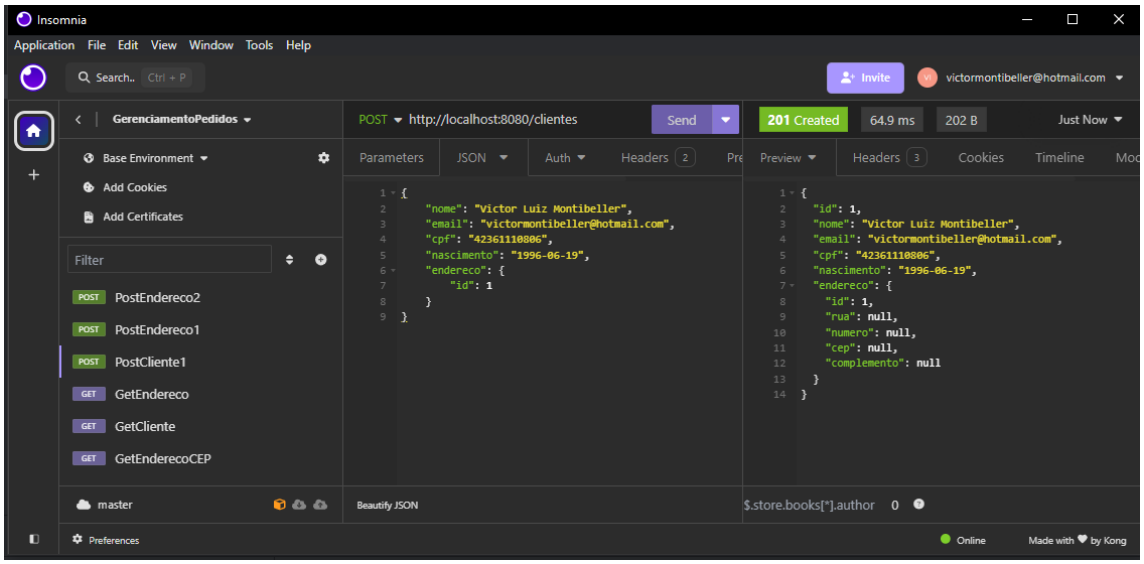

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
Base Environment
Add Cookies
Add Certificates
Filter
POST PostEndereco2
POST PostEndereco1
GET GetEndereco
master
GET http://localhost:8080/enderecos
Send
200 OK
50.1 ms
189 B
Just Now
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
[
{
"id": 1,
"rua": "Rua Prof Benedito Medezzene",
"numero": "137",
"cep": "09845-320",
"complemento": "Casa 1"
},
{
"id": 2,
"rua": "Rua das Rosas",
"numero": "123",
"cep": "01000-000",
"complemento": "Casa 1"
}
]
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong

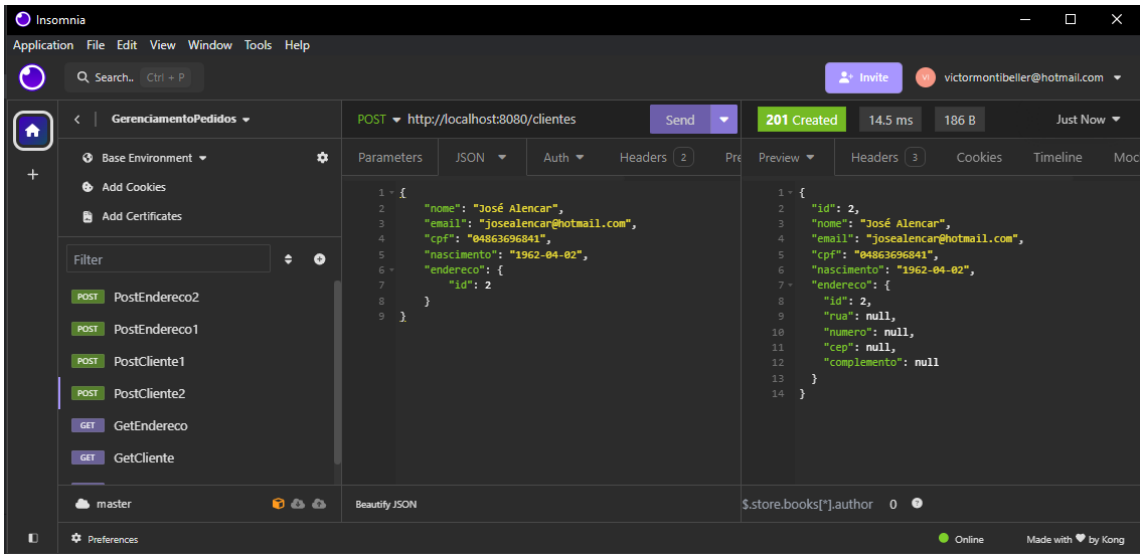

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
Base Environment
Add Cookies
Add Certificates
Filter
POST PostEndereco2
POST PostEndereco1
GET GetEndereco
GET GetEnderecoCEP
master
GET http://localhost:8080/enderecos/cep/010
Send
200 OK
274 ms
86 B
Just Now
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"id": 2,
"rua": "Rua das Rosas",
"numero": "123",
"cep": "01000-000",
"complemento": "Casa 1"
}
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong

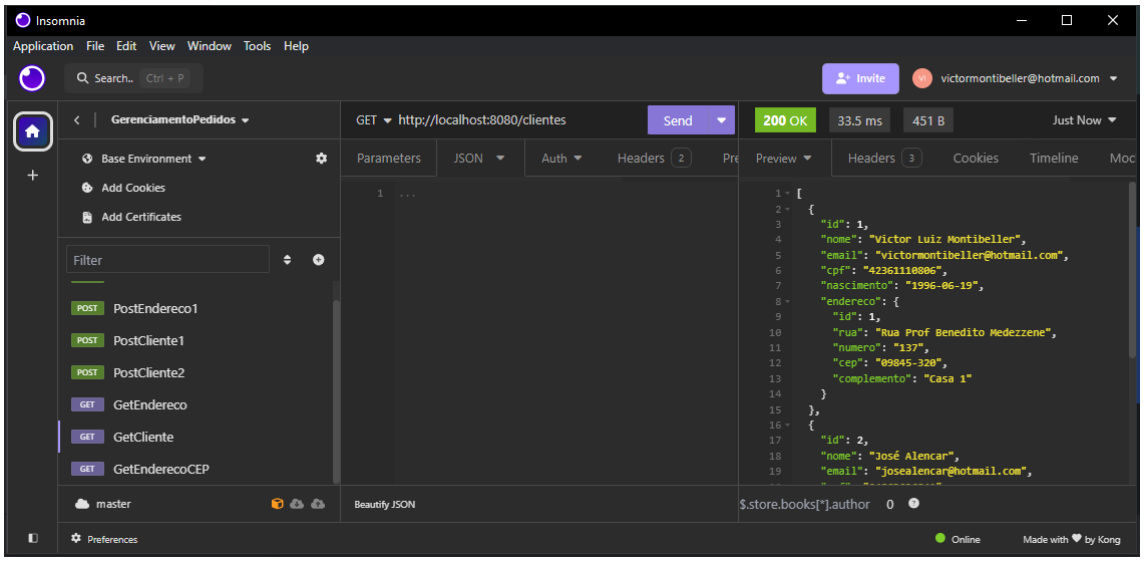
Clientes:

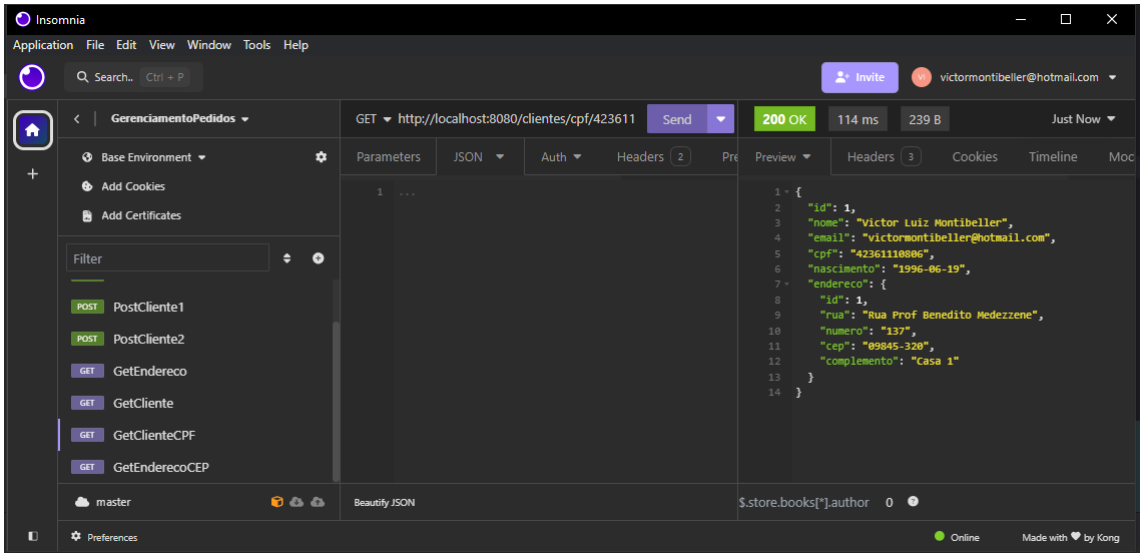

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
GET http://localhost:8080/clientes
Send
404 Not Found
124 ms
110 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"timestamp": "2024-05-21T02:32:05.430+00:00",
"message": "Nenhum cliente encontrado.",
"details": "uri=/clientes"
}
Filter
POST PostEndereco2
POST PostEndereco1
GET GetEndereco
GET GetCliente
GET GetEnderecoCEP
master
Beautify JSON
$.store.books[*].author
Preferences
Online
Made with by Kong


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8080/clientes
Send
404 Not Found
123 ms
125 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"nome": "Victor Luiz Montibeller",
"email": "victormontibeller@hotmail.com",
"cpf": "42361110806",
"nascimento": "1996-06-19",
"endereco": {
"id": 5
}
}
{
"timestamp": "2024-05-21T02:33:24.527+00:00",
"message": "Endereço não encontrado para este id: 5",
"details": "uri=/clientes"
}
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
GET GetEndereco
GET GetCliente
GET GetEnderecoCEP
master
Beautify JSON
$.store.books[*].author
Preferences
Online
Made with by Kong

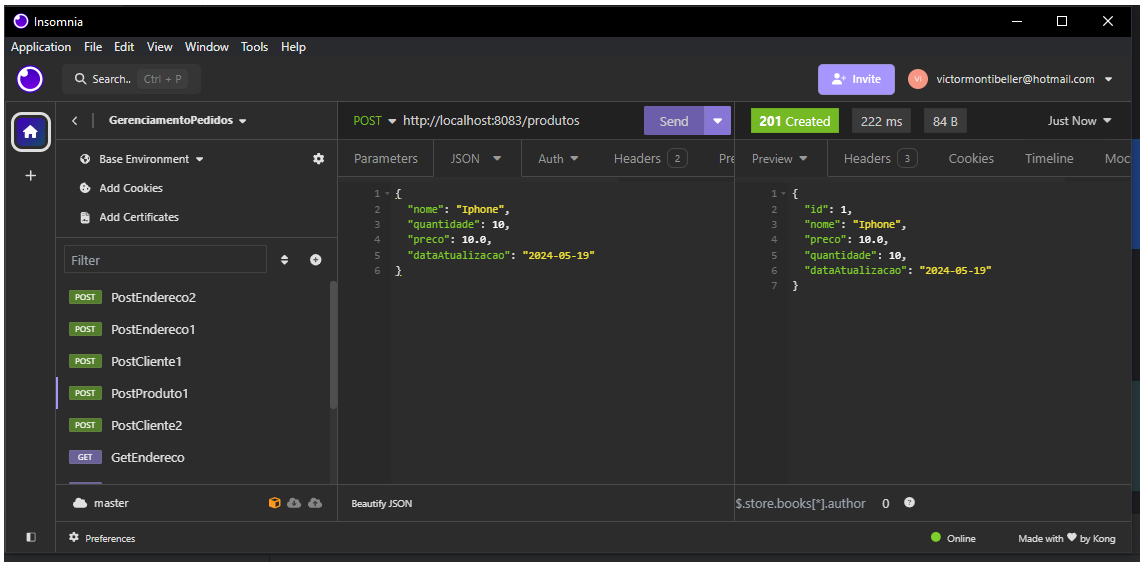

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8080/clientes
Send
201 Created
64.9 ms
202 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"nome": "Victor Luiz Montibeller",
"email": "victormontibeller@hotmail.com",
"cpf": "42361110806",
"nascimento": "1996-06-19",
"endereco": {
"id": 1
}
}
{
"id": 1,
"nome": "Victor Luiz Montibeller",
"email": "victormontibeller@hotmail.com",
"cpf": "42361110806",
"nascimento": "1996-06-19",
"endereco": {
"id": 1,
"rua": null,
"numero": null,
"cep": null,
"complemento": null
}
}
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
GET GetEndereco
GET GetCliente
GET GetEnderecoCEP
master
Beautify JSON
$.store.books[*].author
Preferences
Online
Made with by Kong

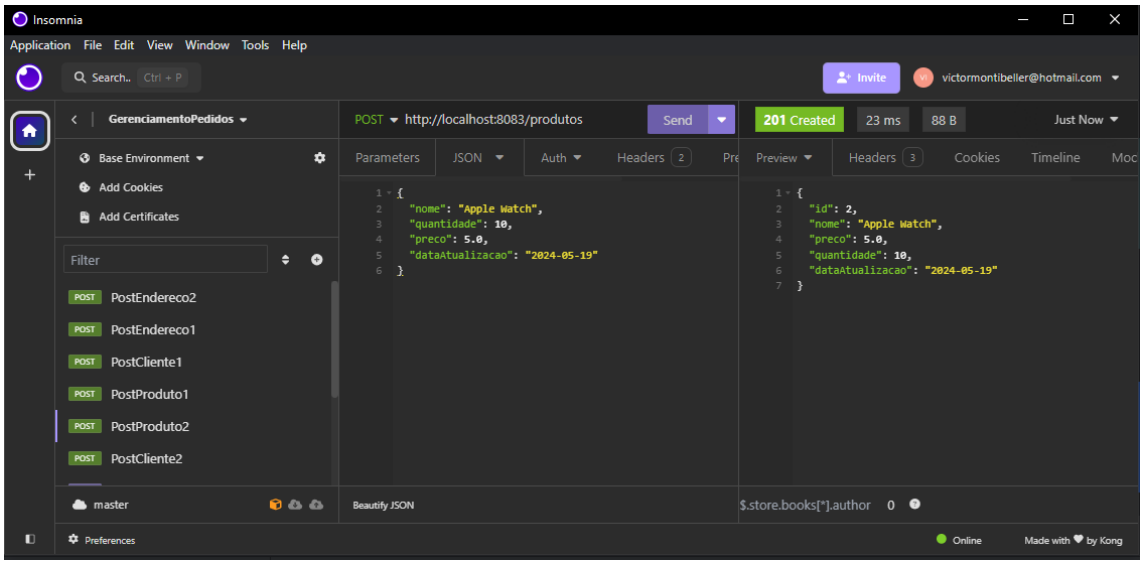

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8080/clientes
Send
201 Created
14.5 ms
186 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"nome": "José Alencar",
"email": "josealencar@hotmail.com",
"cpf": "04863696841",
"nascimento": "1962-04-02",
"endereco": {
"id": 2
}
}
{
"id": 2,
"nome": "José Alencar",
"email": "josealencar@hotmail.com",
"cpf": "04863696841",
"nascimento": "1962-04-02",
"endereco": {
"id": 2,
"rua": null,
"numero": null,
"cep": null,
"complemento": null
}
}
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
POST PostCliente2
GET GetEndereco
GET GetCliente
master
Beautify JSON
$.store.books[*].author 0
Preferences
Online
Made with by Kong

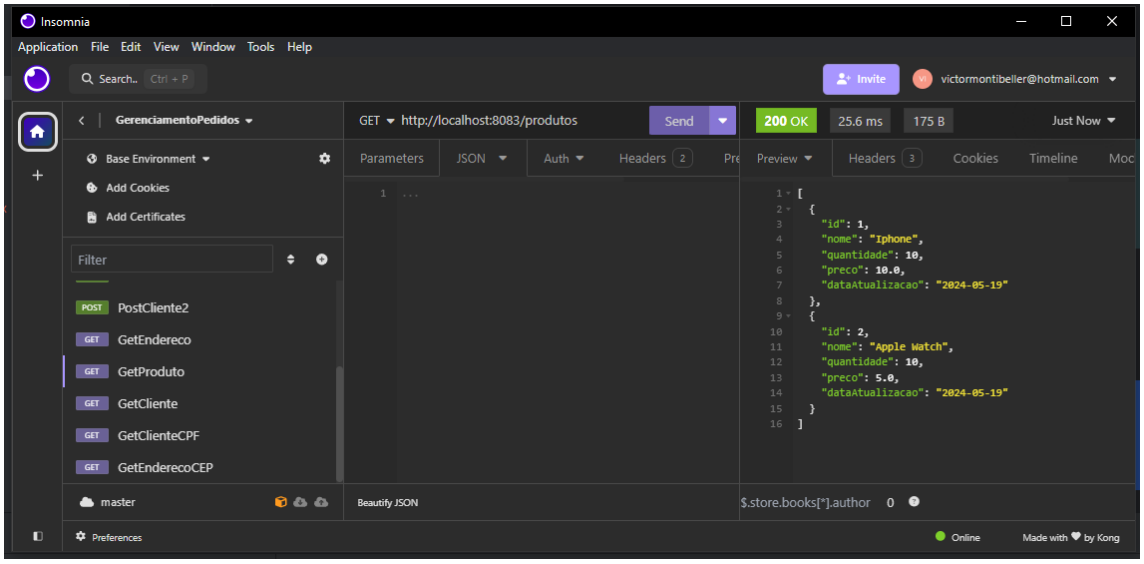

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
GET http://localhost:8080/clientes
Send
200 OK
33.5 ms
451 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
[
{
"id": 1,
"nome": "Victor Luiz Montibeller",
"email": "victormontibeller@hotmail.com",
"cpf": "42361110806",
"nascimento": "1996-06-19",
"endereco": {
"id": 1,
"rua": "Rua Prof Benedito Medezzene",
"numero": "137",
"cep": "09845-320",
"complemento": "Casa 1"
}
},
{
"id": 2,
"nome": "José Alencar",
"email": "josealencar@hotmail.com",
Filter
POST PostEndereco1
POST PostCliente1
POST PostCliente2
GET GetEndereco
GET GetCliente
GET GetEnderecoCEP
master
Beautify JSON
$.store.books[*].author 0
Preferences
Online
Made with by Kong


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
GET http://localhost:8080/clientes/cpf/423611
Send
200 OK
114 ms
239 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
{
"id": 1,
"nome": "Victor Luiz Montibeller",
"email": "victormontibeller@hotmail.com",
"cpf": "42361110806",
"nascimento": "1996-06-19",
"endereco": {
"id": 1,
"rua": "Rua Prof Benedito Medezzene",
"numero": "137",
"cep": "09845-320",
"complemento": "Casa 1"
}
}
Filter
POST PostCliente1
POST PostCliente2
GET GetEndereco
GET GetCliente
GET GetClienteCPF
GET GetEnderecoCEP
master
Beautify JSON
$.store.books[*].author 0
Preferences
Online
Made with by Kong

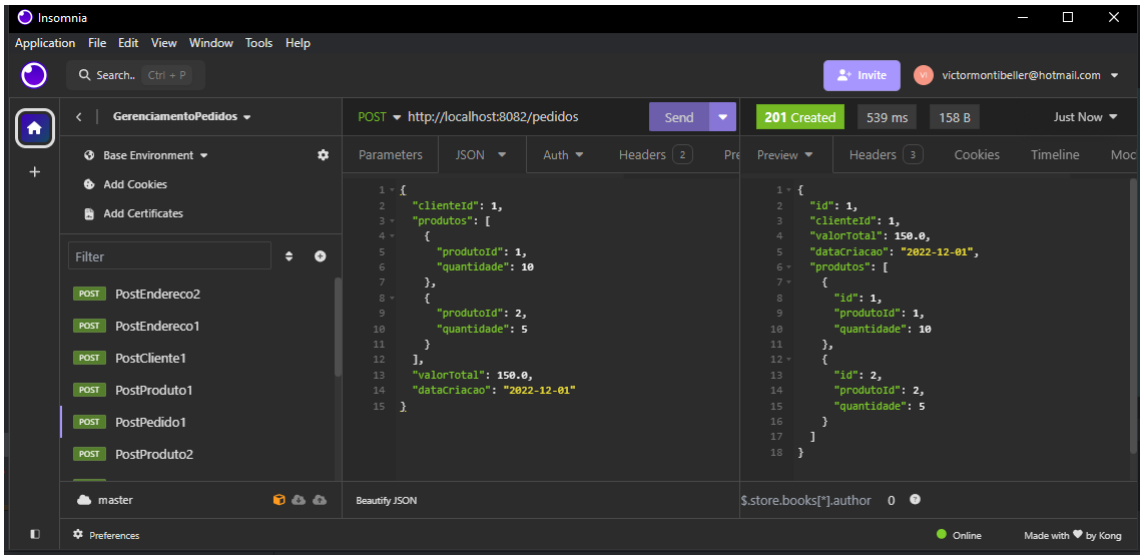
Produtos:

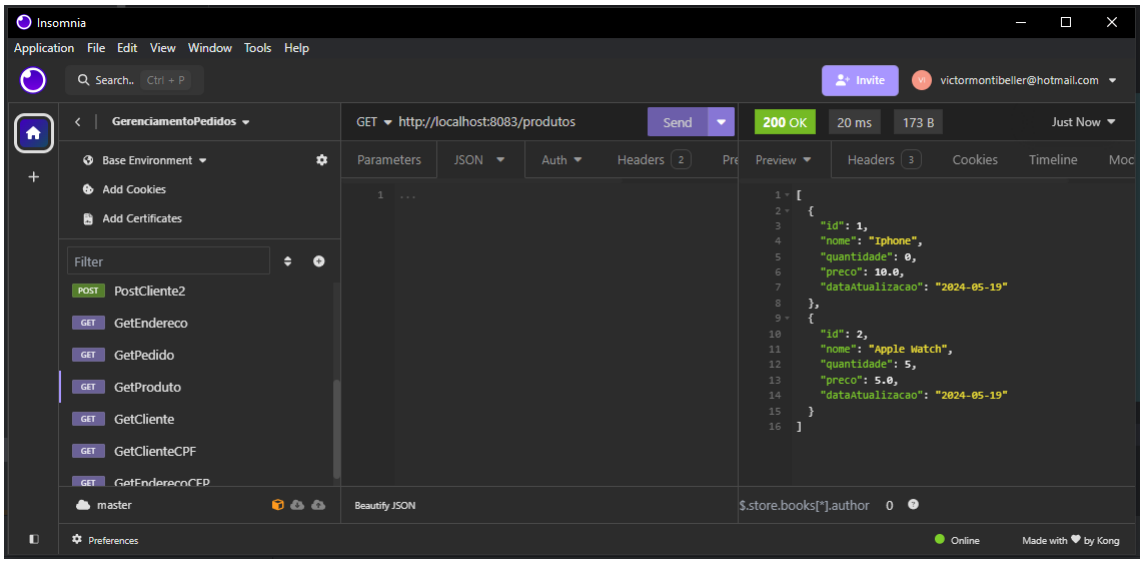

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8083/produtos
Send
201 Created
222 ms
84 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters JSON Auth Headers 2
Preview Headers 3 Cookies Timeline
{
"nome": "Iphone",
"quantidade": 10,
"preco": 10.0,
"dataAtualizacao": "2024-05-19"
}
{
"id": 1,
"nome": "Iphone",
"preco": 10.0,
"quantidade": 10,
"dataAtualizacao": "2024-05-19"
}
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
POST PostProduto1
POST PostCliente2
GET GetEndereco
master
Beautify JSON
$.store.books[*].author 0
Preferences
Online
Made with by Kong

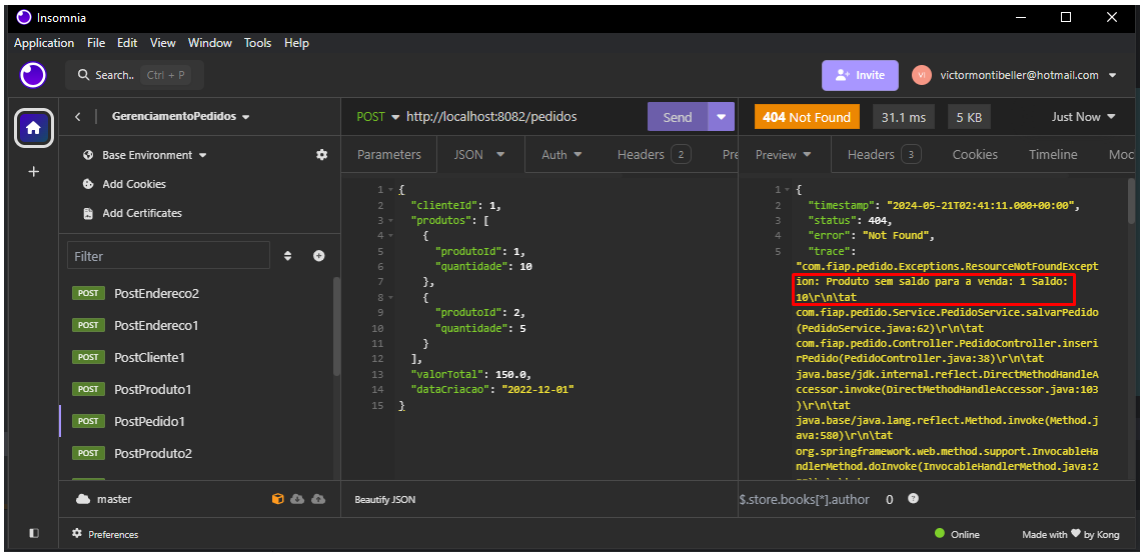

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8083/produtos
Send
201 Created
23 ms
88 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters JSON Auth Headers 2
Preview Headers 3 Cookies Timeline
{
"nome": "Apple Watch",
"quantidade": 10,
"preco": 5.0,
"dataAtualizacao": "2024-05-19"
}
{
"id": 2,
"nome": "Apple Watch",
"preco": 5.0,
"quantidade": 10,
"dataAtualizacao": "2024-05-19"
}
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
POST PostProduto1
POST PostProduto2
POST PostCliente2
master
Beautify JSON
$.store.books[*].author 0
Preferences
Online
Made with by Kong

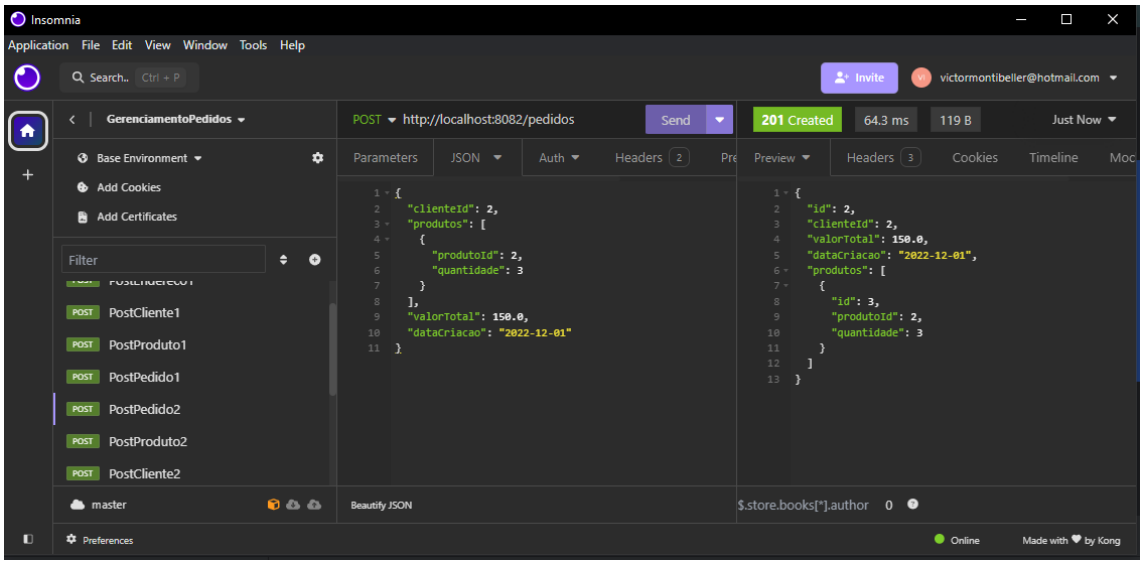

Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
GET http://localhost:8083/produtos
Send
200 OK
25.6 ms
175 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters JSON Auth Headers 2
Preview Headers 3 Cookies Timeline
[
{
"id": 1,
"nome": "Iphone",
"quantidade": 10,
"preco": 10.0,
"dataAtualizacao": "2024-05-19"
},
{
"id": 2,
"nome": "Apple Watch",
"quantidade": 10,
"preco": 5.0,
"dataAtualizacao": "2024-05-19"
}
]
Filter
POST PostCliente2
GET GetEndereco
GET GetProduto
GET GetCliente
GET GetClienteCPF
GET GetEnderecoCEP
master
Beautify JSON
$.store.books[*].author 0
Preferences
Online
Made with by Kong

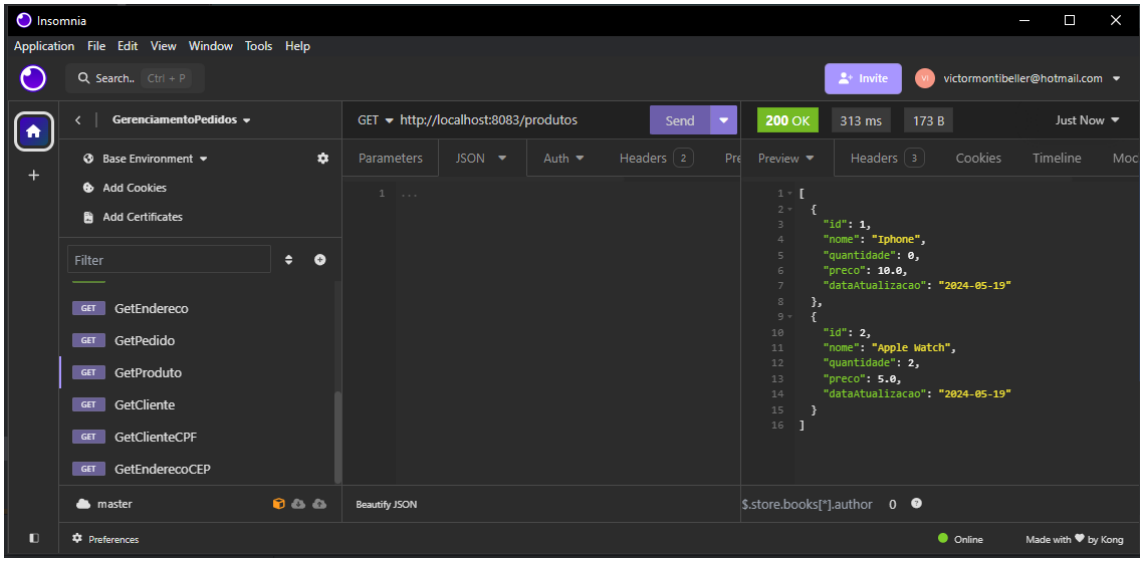
Pedidos:


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8082/pedidos
Send
201 Created
539 ms
158 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
POST PostProduto1
POST PostPedido1
POST PostProduto2
{
"clienteId": 1,
"produtos": [
{
"produtoId": 1,
"quantidade": 10
},
{
"produtoId": 2,
"quantidade": 5
}
],
"valorTotal": 150.0,
"dataCriacao": "2022-12-01"
}
{
"id": 1,
"clienteId": 1,
"valorTotal": 150.0,
"dataCriacao": "2022-12-01",
"produtos": [
{
"id": 1,
"produtoId": 1,
"quantidade": 10
},
{
"id": 2,
"produtoId": 2,
"quantidade": 5
}
]
}
master
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
GET http://localhost:8083/produtos
Send
200 OK
20 ms
173 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
Filter
POST PostCliente2
GET GetEndereco
GET GetPedido
GET GetProduto
GET GetCliente
GET GetClienteCPF
[
{
"id": 1,
"nome": "Iphone",
"quantidade": 0,
"preco": 10.0,
"dataAtualizacao": "2024-05-19"
},
{
"id": 2,
"nome": "Apple Watch",
"quantidade": 5,
"preco": 5.0,
"dataAtualizacao": "2024-05-19"
}
]
master
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8082/pedidos
Send
404 Not Found
31.1 ms
5 KB
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
Filter
POST PostEndereco2
POST PostEndereco1
POST PostCliente1
POST PostProduto1
POST PostPedido1
POST PostProduto2
{
"clienteId": 1,
"produtos": [
{
"produtoId": 1,
"quantidade": 10
},
{
"produtoId": 2,
"quantidade": 5
}
],
"valorTotal": 150.0,
"dataCriacao": "2022-12-01"
}
{
"timestamp": "2024-05-21T02:41:11.000+00:00",
"status": 404,
"error": "Not Found",
"trace":
"com.fiap.pedido.Exceptions.ResourceNotFoundException: Produto sem saldo para a venda: 1 Saldo: 10\r\n\tat
com.fiap.pedido.Service.PedidoService.salvarPedido(PedidoService.java:62)\r\n\tat
com.fiap.pedido.Controller.PedidoController.inserirPedido(PedidoController.java:38)\r\n\tat
java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:103)\r\n\tat
java.base/java.lang.reflect.Method.invoke(Method.java:580)\r\n\tat
org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:2
master
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
POST http://localhost:8082/pedidos
Send
201 Created
64.3 ms
119 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
Filter
POST PostCliente1
POST PostProduto1
POST PostPedido1
POST PostPedido2
POST PostProduto2
POST PostCliente2
{
"clienteId": 2,
"produtos": [
{
"produtoId": 2,
"quantidade": 3
}
],
"valorTotal": 150.0,
"dataCriacao": "2022-12-01"
}
{
"id": 2,
"clienteId": 2,
"valorTotal": 150.0,
"dataCriacao": "2022-12-01",
"produtos": [
{
"id": 3,
"produtoId": 2,
"quantidade": 3
}
]
}
master
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Invite
victormontibeller@hotmail.com
GerenciamentoPedidos
GET http://localhost:8083/produtos
Send
200 OK
313 ms
173 B
Just Now
Base Environment
Add Cookies
Add Certificates
Parameters
JSON
Auth
Headers 2
Preview
Headers 3
Cookies
Timeline
Filter
GET GetEndereco
GET GetPedido
GET GetProduto
GET GetCliente
GET GetClienteCPF
GET GetEnderecoCEP
[
{
"id": 1,
"nome": "Iphone",
"quantidade": 0,
"preco": 10.0,
"dataAtualizacao": "2024-05-19"
},
{
"id": 2,
"nome": "Apple Watch",
"quantidade": 2,
"preco": 5.0,
"dataAtualizacao": "2024-05-19"
}
]
master
Beautify JSON
$.store.books[*].author
0
Preferences
Online
Made with by Kong

Considerações Finais

O desenvolvimento da API de Gerenciamento de Pedidos representou uma jornada empolgante em direção à construção de uma solução robusta e de alta qualidade. Durante este projeto, demos destaque à arquitetura limpa, seguindo os princípios da Clean Architecture e MicroServiços , que nos permitiram criar um código modular, escalável e de fácil manutenção.

Durante o desenvolvimento da API, exploramos diversas tecnologias e ferramentas, incluindo Spring, JPA, Batch, Cloud Stream, Hibernate e Mockito, e aplicamos práticas de engenharia de software, como testes unitários, testes de integração e análise de cobertura de testes.

Os testes desempenharam um papel crucial em nosso processo de desenvolvimento, com ênfase em testes unitários e de integração. Utilizamos ferramentas como JUnit e Mockito para garantir a qualidade e confiabilidade de nossa aplicação. Além disso, a integração com Docker facilitou a distribuição e execução da API em diferentes ambientes de desenvolvimento e produção.

Nossa jornada não foi apenas uma oportunidade de aplicar conceitos teóricos, mas também um aprendizado contínuo e colaborativo. A troca de conhecimentos e a resolução de desafios em equipe fortaleceram nossa compreensão e habilidades em engenharia de software.

À medida que concluímos este projeto, reconhecemos que sempre há espaço para melhorias e refinamentos. No entanto, estamos orgulhosos do que alcançamos e confiantes de que nossa API oferece uma base sólida para futuras iterações e extensões.

Agradecemos à FIAP por nos proporcionar esta oportunidade de aprendizado e crescimento, e aos nossos colegas e professores por seu apoio e orientação ao longo do curso. Estamos ansiosos para aplicar os conhecimentos adquiridos neste projeto em nossas jornadas futuras no desenvolvimento de software.