

**Documentação de API**

**Gerenciador de Restaurante ou Lanchonete**

# Introdução

Software desenvolvido para concorrer a vaga de **Desenvolvedor Back-end** na empresa Maida.health.

Este software consiste na criação de uma api REST full, utilizando do Spring boot, Java 18, Spring Security com Jwt e Junit. O seu objetivo é possibilitar o gerenciamento de um estabelecimento de produtos consumíveis. Podendo ser uma lanchonete ou restaurante.

Ela é basicamente uma central de pedidos, que permite o cliente realizar seus pedidos e acompanhar seu andamento. E permitirá o gestor ou responsável pelo estabelecimento, gerenciar os pedidos, e produtos ofertados.

# Módulos Existentes

* Usuários - Este modulo conterá endpoints para o gerenciamento de usuários do estabelecimento, podendo ser clientes ou gestores

Regras deste modulo:

* + Conterá apenas um usuário que atuara como gestor do estabelecimento, o sistema não permitira o cadastro mais gestores.
  + Os clientes poderão se cadastrar livremente por este modulo. Pelo endpoint /restaurante/user/create
  + Os clientes serão controlados por seus e-mails, podendo ter apenas um usuário por e-mail.
  + Para alteração de usuário, o cliente deverá solicitar ao responsável pelo estabelecimento (Gestor)
  + O gestor poderá alterar as informações dos usuários.
  + O gestor poderá realizar busca por usuários através de sua identificação.
  + O gestor poderá carregar todos os usuários paginados.
  + Os dois tipos de usuários terão acesso livre ao endpoint de login. Necessitaram passar nele seu e-mail e sua senha para a obtenção token de autenticação de seu acesso. Com ele poderão navegar pelos demais módulos da API, conforme seu nível de permissão.
* Clientes – Este módulo conterá endpoints para gerenciamento de clientes. Nele os usuários cadastrados como cliente deverão complementar suas informações (sendo obrigatórios) para realização de pedidos futuros.

Regras deste modulo:

* + Após o usuário estiver logado, ele executara a criação de um novo cliente para seu usuário, através do endpoint /restaurante/cliente/create
  + O Usuário do tipo Login poderá alterar as informações deste modulo. Mas estará impedido de realizar qualquer tipo de consulta.
  + O Usuário do tipo Gestor poderá realizar consultas para ver todos os clientes paginados ou consulta por uma identificação específica.
* Endereços - Este módulo permite ao cliente depois de estar devidamente cadastrado e atualizada suas informações, terá acesso a este modulo. Que permite cadastrar seus endereços. Neste modulo o usuário do tipo cliente terá acesso total. Podendo criar, alterar, pesquisar, carregar seus endereços paginados e removê-los.
* Produtos – Este modulo permitirá ao gestor gerenciar seus produtos podendo criar novos produtos, alterá-los, carregá-los de forma paginada e pesquisar produtos específicos através de sua identificação.
* Pedidos – Este modulo permitirá tanto o gestor gerenciar os pedidos lançados pelos clientes, como também permitirá o cliente criar pedidos, acompanhá-los ou cancelá-los se desistir antes de serem atendidos. Neste modulo ambos terão acesso em todos os endpoints, O sistema trabalhará internamente para entregar as informações conforme a permissão de cada um.

# Endpoints e permissões

Assim se comportaram os endpoints em relação as permissões: tipos de usuários (CLIENTE E GESTOR)

1. LOGIN => PERMITIDO A TODOS
2. USUARIO

CRIAR USUARIO => PERMITIDO A TODOS

ALTERAR USUARIO => GESTOR

BUSCAR USUARIO PELA IDENFICAÇÃO => GESTOR

1. CLIENTE

LISTAR CLIENTES => GESTOR

BUSCAR CLIENTE => GESTOR

CRIAR CLIENTE => CLIENTE

ALTERAR CLIENTE => CLIENTE

1. PEDIDOS

ALTERAR PEDIDOS => PERMITIDO A TODOS

LISTAR PEDIDOS => PERMITIDO A TODOS

BUSCAR PEDIDOS => PERMITIDO A TODOS

CRIAR PEDIDOS => CLIENTE

1. PEDIDOS

LISTAR PRODUTOS => GESTOR

BUSCAR PRODUTOS => GESTOR

ALTERAR PRODUTOS => GESTOR

CRIAR PRODUTOS => GESTOR

# Processo de Desenvolvimento

O processo de desenvolvimento dessa aplicação durou 7 dias corridos, dedicando cerca de 6 a 8 horas por dia. A dificuldade dela para mim foi média:

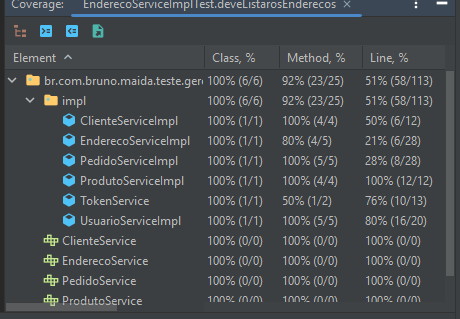
* + Iniciei o processo de desenvolvimento pelas configurações de segurança utilizando o Spring Security e jwt. Neste trabalho utilizei dois dias para encontrar a melhor forma de trabalhar estas ferramentas em meu projeto.
  + Após comecei a criação projeto definindo a arquitetura e também os endpoints que deveriam ser criados e a forma de suas distribuições, com isso usei mais dois dias.
  + Assim que concluído desenvolvimento da aplicação, iniciei a documentação do Swagger, para isso utilizei o Spring Doc para documentar os endpoints. Utilizei esta ferramenta por ela ter mais liberdade na sua manipulação. Nesta tarefa utilizei um dia.
  + Após a finalização da documentação, realizei algumas melhorias de código e também correções de bugs. Utilizei um dia para realizar este trabalho.
  + E por fim iniciei os testes com Junit e Mockito, construí todos os casos de testes dos Services. E alguns executaram com êxito, já outros tive problemas com a execução do jwt nos testes. Onde pedia o contexto do usuário, emitiram erros. Embora conheça o Junit. Não cheguei passar por estes problemas com o jwt anteriormente, e não terei tempo hábil para achar a correção. Mas deixo como ponto de melhoria em meu projeto.

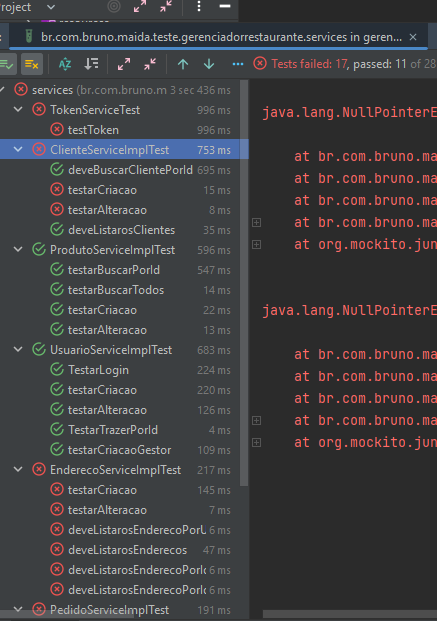
Desta forma senti apenas dificuldades neste contexto de teste com Junit e Jwt. E assim, foi minha experiência com esta API.

No Readme do projeto está descrito mais algumas coisas sobre o projeto.

# Junit do projeto

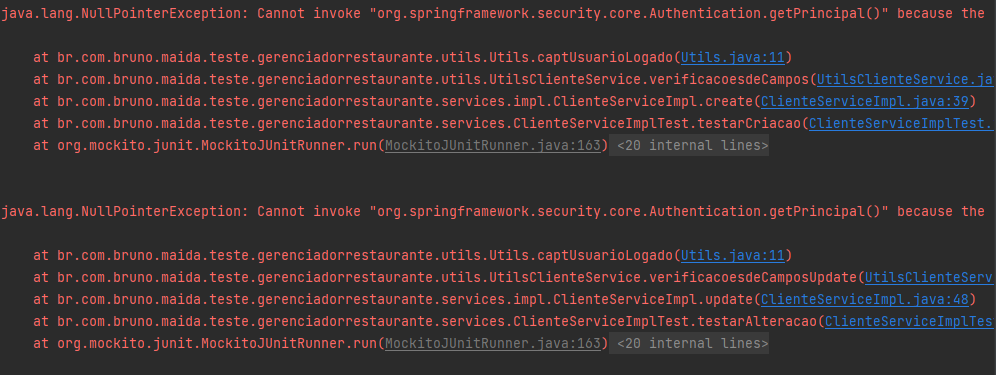
Realizei os teste unitarios com o junit. Ficaram assim os resultados, atingindo 100% das classes, 92% dos methodos e 51% das linhas

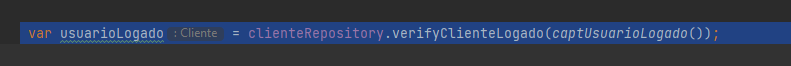




As falhas aconteceram por conta do jwt, Não consegui realizar os testes nos métodos que verificam o usuário logado para executar a ação. Isto coloco como melhoria em meu projeto. Não tive tempo hábil para achar a solução, para correção destes teste.

Segue o erro tomado e a linha motivo.







Este método captura o usuario atual, mas no momento dos teste ele não encontra este contexto.