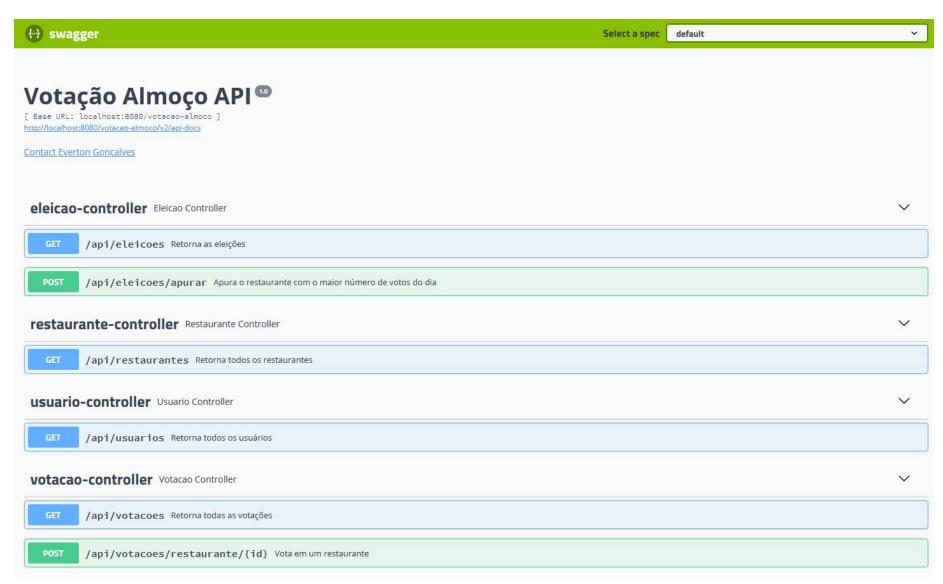
Votação Almoço API

Informações de acesso

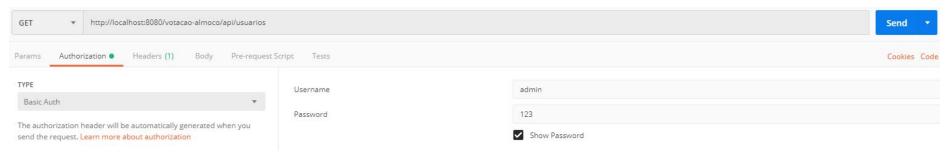
Usuário	Senha	Permissão de acesso
admin	/api/e	Todas
apurador		/api/eleicoes/*
faminto_1		
faminto_2		
faminto_3		
faminto_4		
faminto_5		/api/votacoes/*
faminto_6		/api/votacoes/
faminto_7		
faminto_8		
faminto_9		
faminto_10		



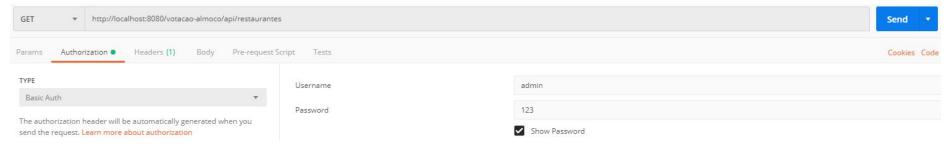
Acesso ao Swagger: http://localhost:8080/votacao-almoco/swagger-ui.html

Utilização da ferramenta Postman

Usuários



Restaurantes



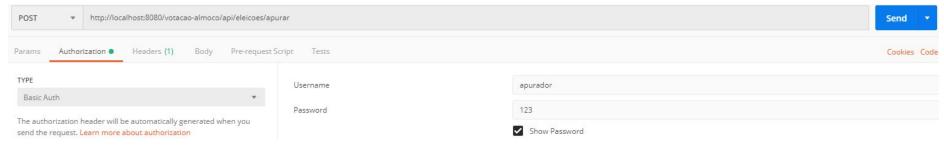
Votações



Realizar votação em um restaurante



Apurar restaurante vencedor



Eleições



Informações

- O horário limite para a votação é configurado na propriedade **votacao.horaLimite** localizada no arquivo application.properties da aplicação e leva o formato HH:mm.
- Ao iniciar a aplicação, a base de dados é criada e o script localizado import.sql é executado automaticamente.

Execução da aplicação

Requisitos:

- Apache Maven 3.3.X
- Java 1.8

Iniciar a aplicação via Spring:

• mvn spring-boot:run

Iniciar a aplicação via comando java:

- mvn clean package
- java -jar target/votacao-almoco-api-1.0.0.jar

Tecnologias utilizadas

- Spring Boot: Framework de código aberto que facilita o desenvolvimento e configuração de APIs.
- H2: Sistema de gerenciamento de banco de dados relacional escrito em Java.
- Maven: Ferramenta para automação de builds e gerenciamento das dependências do projeto.
- JUnit: Framework de código aberto para implementação de testes de unidade.
- Lombok: Framework que provê implementações de código com o objetivo de auxiliar na produtividade e redução do mesmo.
- Swagger: Framework para documentar APIs Rest.

Pontos de destaque

- Utilização e configuração em memória do banco H2.
- Utilização do Spring Data para a persistência de dados.
- Delegação do controle transacional ao Spring na camada Service.
- Utilização de autenticação básica do Spring Security com usuários em memória e permissões para controle de acesso para as rotas.
- Tratamento de erros através da classe BaseExceptionHandler e apresentação customizada das mensagens através da classe
 ErrorDetails.
- Padronização de mensagens com suporte a internacionalização utilizando a classe utilitária MessageSource do Spring.
- Utilização do design pattern Bridge para separação de cada regra de negócio com o objetivo de reduzir a quantidade de código da camada Service e facilitar os testes unitários.
- Utilização do design pattern Adapter para converter um objeto para que outro o entenda.
- Utilização do design pattern Builder para a criação de objetos complexos através do Lombok.
- Utilização de generics na estrutura de validators.

Sugestão de melhoria

- Testes de integração.
- Implementação de programação orientada a aspectos (Spring AOP) para logs do sistema.
- Criação de arquitetura de microserviços desmembrando componentes mínimos e independentes (usuários e restaurantes).