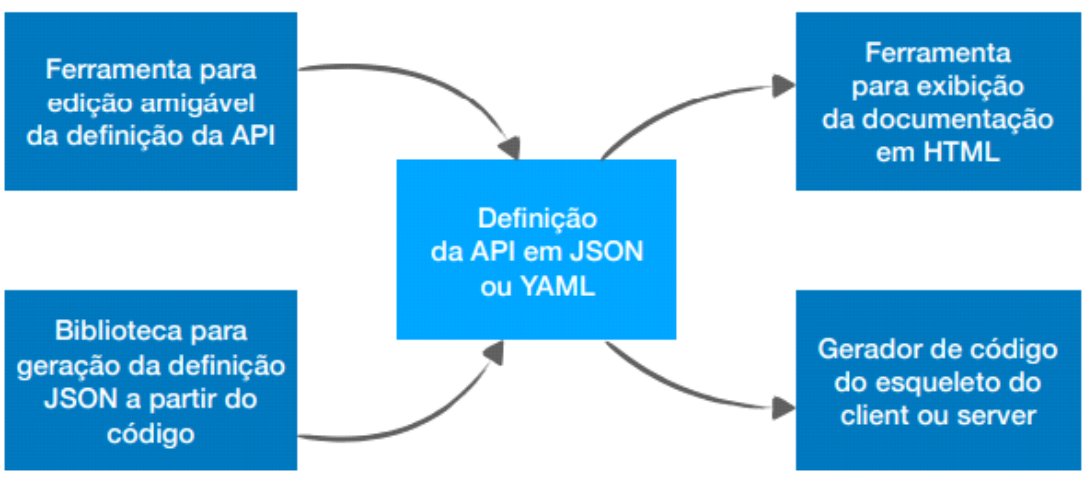
18.1. Introdução à documentação de REST APIs

domingo, 9 de abril de 2023 15:03

18.2. Conhecendo a OpenAPI (antes conhecida como Swagger)

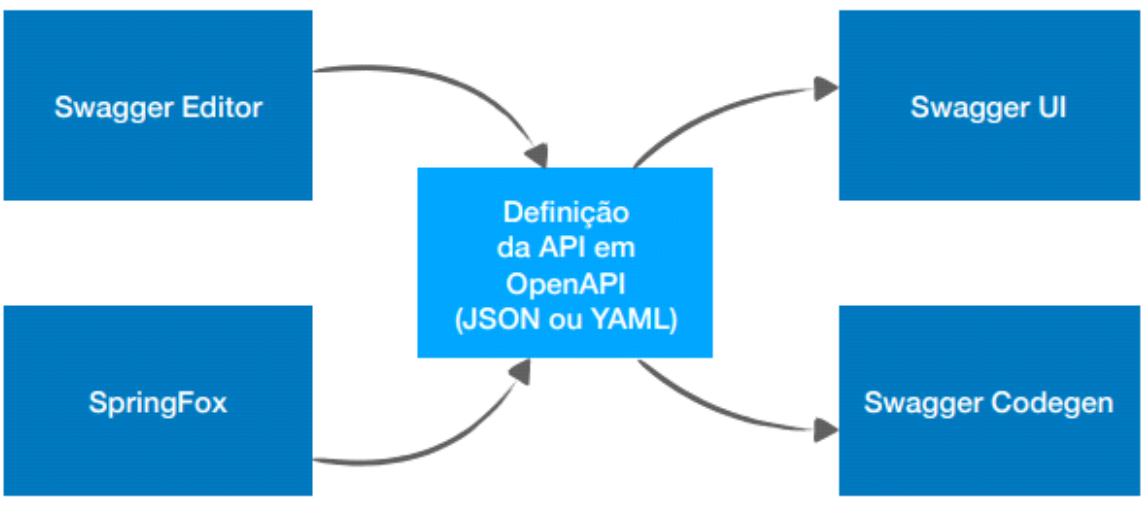
domingo, 9 de abril de 2023 15:08

- Padronização para descrever o funcionamento Rest Apis
  - Arquivo JSON ou YAML com toda a definição da api
  - Descreve endpoints, parametros de entrada, modelos de representação de entrada e saida, codigos http, informações gerais sobre a api
  - Pode ser escrito e lido por pessoas ou ferramentas
- Por que usar uma especificação?
  - Abre possibilidades de uso de ferramentas que foram criadas em torno da especificação
  - Comunidade que implementa especificações, podem ser pagas ou grautuitas
  - Independente de linguagem
- Utilidade



- Swagger
  - Swagger Specification 2 (até versão 2)
  - Especificação proprietária pertencente a empresa SmartBear até 2015
  - 2015 a SmartBear doou a especificação para a comunidade mudando o nome para OpenApi Specification
  - Mantida desde então pela OpenApi Initiative
  - SmartBean desenvolveu várias ferramentas em torno da OpenApi Specification
  - Ferramentas possui prefixo "Swagger"
  - Swagger é um conjunto de ferramentas que auxiliam quem adota a OpenApiSpecification
  - SmartBean possui ferramentas OpenSourcers e Comeriais
  - As mais conhecidas são da SmartBear

- Ferramentas Swagger da SmartBear



  - Swagger Editor
    - Ajuda a editar arquivos de definição json ou yaml dentro do navegador e mostra uma documentação em tempo real
  - SpringFox
    - Biblioteca java gratuita, scaneia o código e gera o arquivo de definição JSON e gera documentação com Swagger UI
  - Swagger UI
    - Documentação em HTML que é gerado dinamicamente a partir de um arquivo de definição de API. JSON -> Documentação HTML
  - Swagger Codegen
    - Gera código de clientes API

18.3. Gerando a definição OpenAPI em JSON com SpringFox

domingo, 9 de abril de 2023 17:14

Atualizando para o Spring 2.7++, SpringFox 3.0 e Open API 3

Dependência Spring Fox 3

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-boot-starter</artifactId>
  <version>3.0.0</version>
</dependency>
```

Classe de configuração:

```
package com.algaworks.algafood.core.openapi;

import ...

@Configuration
public class SpringFoxConfig {

    @Bean
    public Docket apiDocket(){

        return new Docket(DocumentationType.OAS_30)
                .select()
                .apis(RequestHandlerSelectors.any())
                .build();
    }
}
```

select() retornar builder para selecionar endpoints para serem expostos
apis() quais controladores e endpoints o springfox tem que expor
RequestHandlerSelectors.any como argumento: expõe todos os endpoints

A partir da versão 2.6.0 ++ , uma configuração adicional é necessária para o funcionamento do SpringFox 3.0.0. Essa configuração pode ser feita de duas formas diferentes:

Adição de configuração no application.properties
spring.mvc.pathmatch.matching-strategy=ANT_PATH_MATCHER

Adição de anotação na classe principal
A anotação @EnableWebMvc pode ser adicionada à classe principal da aplicação (AlgafoodApiApplication.java)

Acessando a documentação na versão 3
Utilize a URL http://localhost:8080/v3/api-docs para acessar o JSON de configuração na versão 3.

http://springfox.github.io/springfox/docs/current/

18.4. Gerando a documentação da API em HTML com Swagger UI e SpringFox

domingo, 9 de abril de 2023 18:46

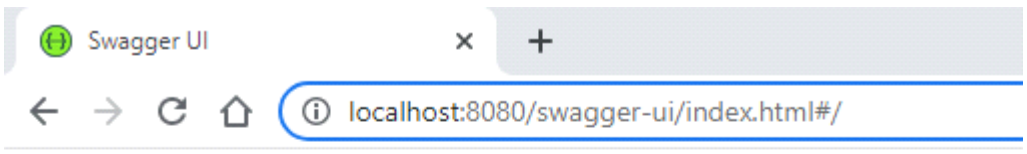
Para SpringFox 3.0 e Open API 3 o próprio SpringFox 3 já traz como dependência transitiva o springfox-swagger-ui

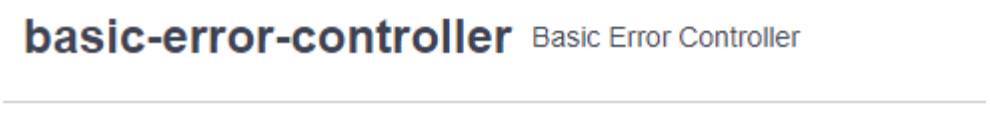
Para acessar a documentação montada com o JSON do SpringFox 3

http://localhost:8080/swagger-ui/index.html

18.5. Selecionando os endpoints da API para gerar a documentação

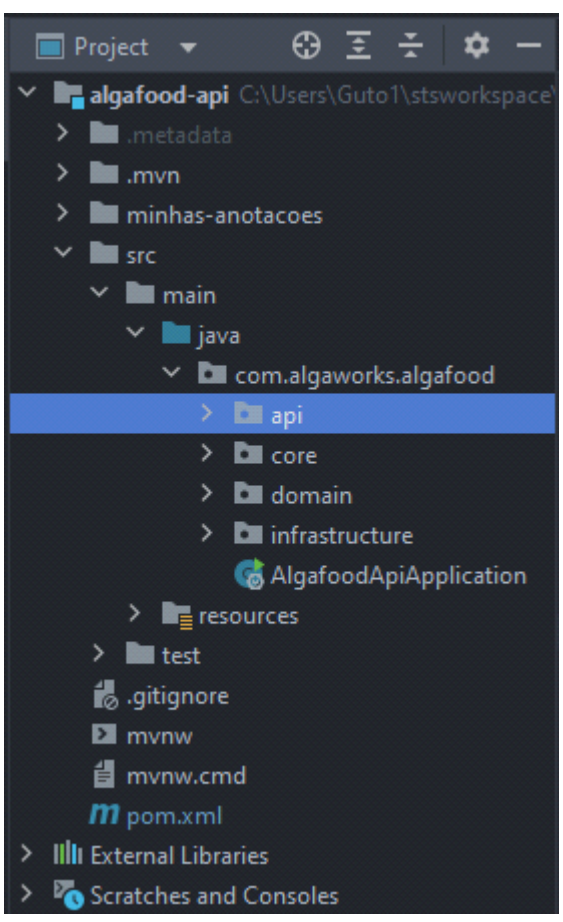
domingo, 9 de abril de 2023 19:05





Para retirar o basic controlador (Spring Boot) podemos filtrar por pacote o que será scaneado pelo SpringFox

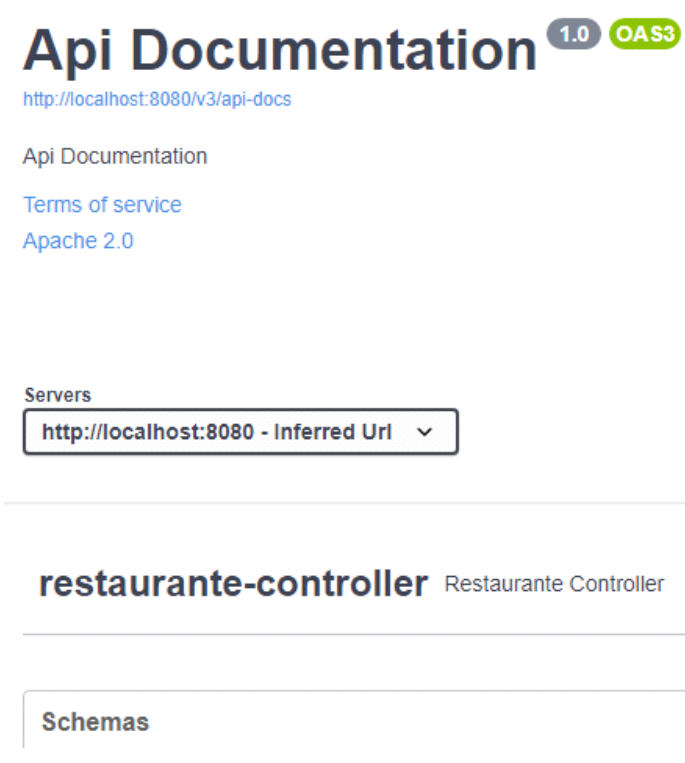
```java
@Configuration
public class SpringFoxConfig {

    @Bean
    public Docket apiDocket(){

        return new Docket(DocumentationType.OAS_30)
                .select()
                //.apis(RequestHandlerSelectors.any())/*Todos os endpointds*/
                .apis(RequestHandlerSelectors.basePackage("com.algaworks.algafood.api"))
                .build();
    }
}
```



Podemos adicionar caminhos para o SpringFox scanear os endpoints

```java
@Configuration
public class SpringFoxConfig {

    @Bean
    public Docket apiDocket(){

        return new Docket(DocumentationType.OAS_30)
                .select()
                //.apis(RequestHandlerSelectors.any())/*Todos os endpointds*/
                .apis(RequestHandlerSelectors.basePackage("com.algaworks.algafood.api"))
//                .paths(PathSelectors.any())/*Caminho padrão*/
                .paths(PathSelectors.ant( antPattern: "/restaurantes/*"))
                .build();
    }
}
```

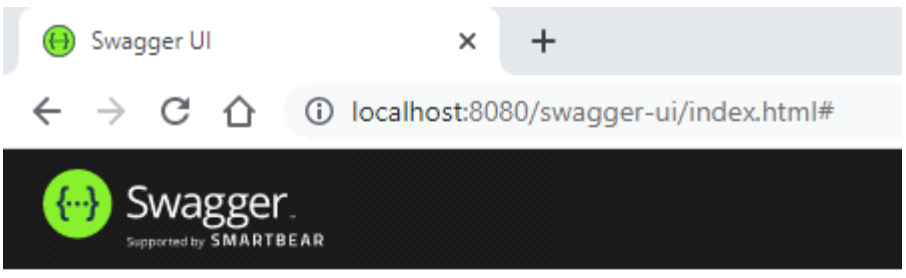
No código acima, o SpringFox irá scanear os endpoints do pacote base e também apenas endpoints com o path definido em PathSelectors

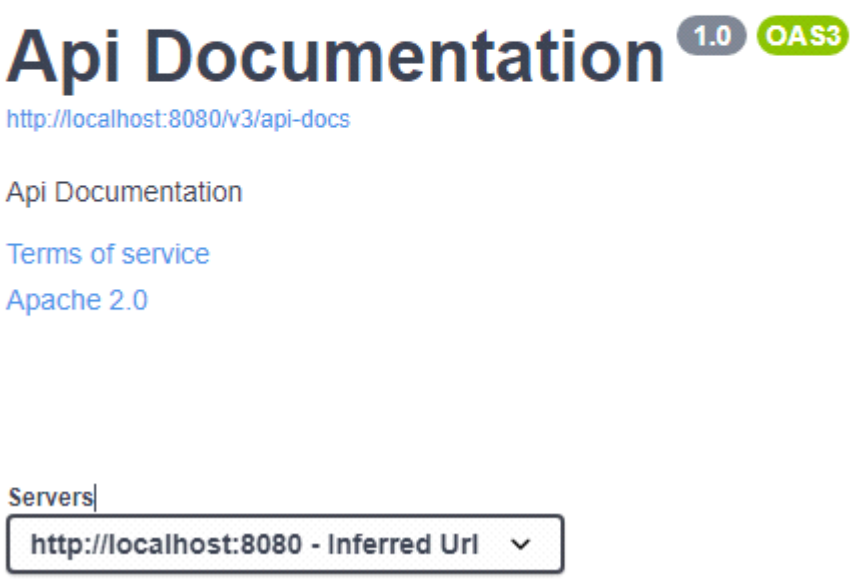


18.6. Descrevendo informações da API na documentação

domingo, 9 de abril de 2023 19:39

- Podemos alterar as informações da documentação padrão para uma customizada criando uma instância de ApiInfo
- Página padrão

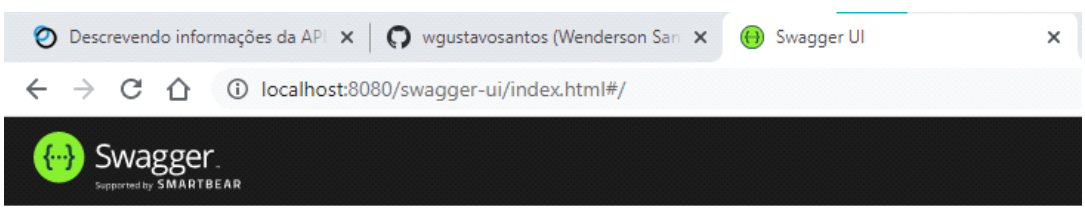


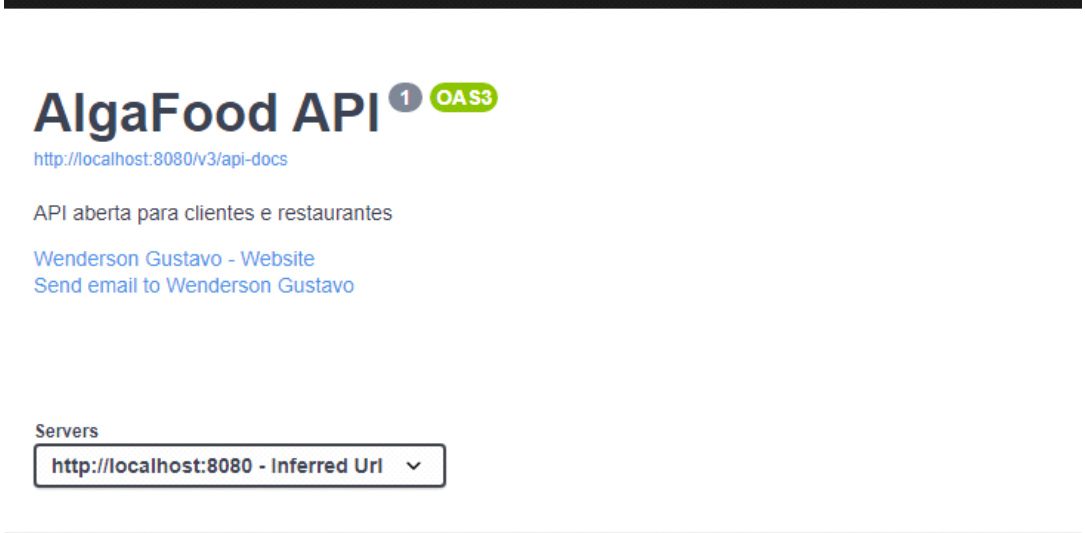


- Implementação

```java
@Configuration
public class SpringFoxConfig {

    @Bean
    public Docket apiDocket(){...}

    1 usage
    public ApiInfo apiInfo(){

        final Contact wenderson_gustavo =
                new Contact( name: "Wenderson Gustavo",
                        url: "https://github.com/wgustavosantos",
                        email: "wgustavo.dev@gmail.com");

        return new ApiInfoBuilder()
                .title("AlgaFood API")
                .description("API aberta para clientes e restaurantes")
                .version("1")
                .contact(wenderson_gustavo)
                .build();

    }
}
```

- Página customizada

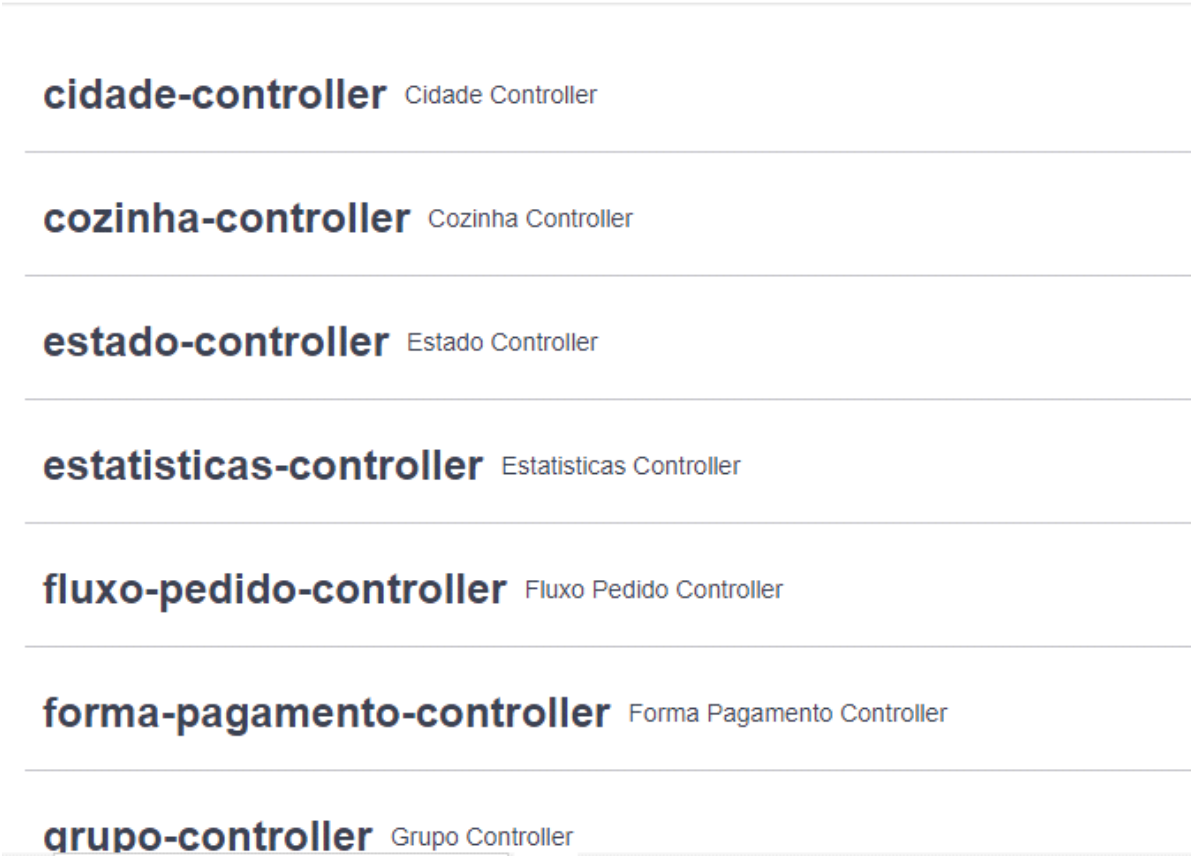




18.7. Descrevendo tags na documentação e associando com controllers

domingo, 9 de abril de 2023 21:15

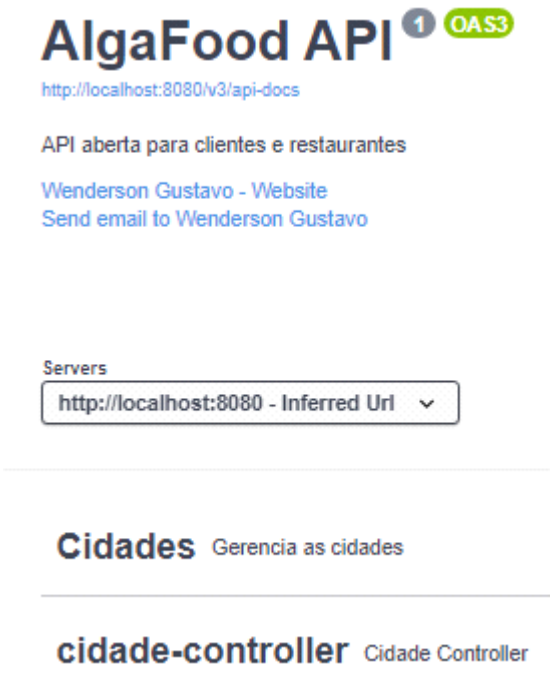
- Cada descrição de controladores na página inicial do Swagger UI 3 é chamada de Tags, e não somente na Swagger UI 3 como na especificação OpenApi.



Podemos criar novas Tags para a documentação adicionando uma instância no builder de Docket na classe de configuração

```
@Bean
public Docket apiDocket() {

    return new Docket(DocumentationType.OAS_30)
            .select()
            //.apis(RequestHandlerSelectors.any())/*Todos os endpointds*/
            .apis(RequestHandlerSelectors.basePackage("com.algaworks.algafood.api"))
            .paths(PathSelectors.any())/*Caminho padrão*/
              .paths(PathSelectors.ant("/restaurantes/*"))
            .build()
            .apiInfo(apiInfo())
            .tags(new Tag( name: "Cidades",  description: "Gerencia as cidades"));
}
```

porém, a tag não está vinculada com nenhum endpoint



Para vincular uma tag a um controlador para ser visualizado na página do Swagger UI, basta adicionar a anotação @Api na classe controladora

```
package com.algaworks.algafood.api.controller;

import ...

@Api(tags = "cidades")
@RestController
@RequestMapping("/cidades")
public class CidadeController {
```

na página


Cidades Gerencia as cidades
GET /cidades listar
POST /cidades adicionar
GET /cidades/{cidadeId} buscar
DELETE /cidades/{cidadeId} deletar
PUT /cidades/{id} atualizar
cozinha-controller Cozinha Controller
estado-controller Estado Controller

18.8. Descrevendo as operações de endpoints na documentação

domingo, 9 de abril de 2023 21:30

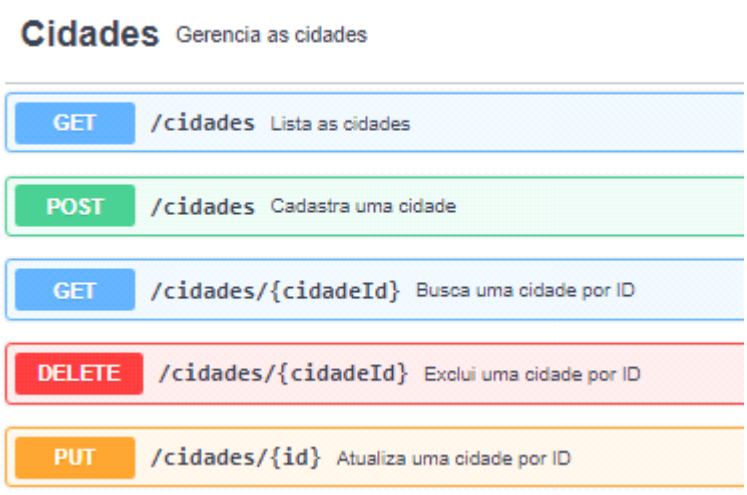
O Swagger UI 3 utiliza o nome dos métodos dos endpoints do controlador para descrever as operações dos endpoints na api documentada

Para mudar o nome das operações dos endpoints, basta adicionar a anotação @ApiOperation em cada método do controlador desejado

- Controlador de cidades na api

```
@ApiOperation("Cadastra uma cidade")
@PostMapping
public ResponseEntity<CidadeDTO> adicionar(
@ApiOperation("Lista as cidades")
@GetMapping
public ResponseEntity<List<CidadeDTO>> list

@ApiOperation("Busca uma cidade por ID")
@GetMapping("/{cidadeId}")
public CidadeDTO buscar(@PathVariable Long

@ApiOperation("Atualiza uma cidade por ID")
@PutMapping("/{id}")
public CidadeDTO atualizar(@RequestBody @Va

@ApiOperation("Exclui uma cidade por ID")
@DeleteMapping("/{cidadeId}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deletar(@PathVariable Long cida
```

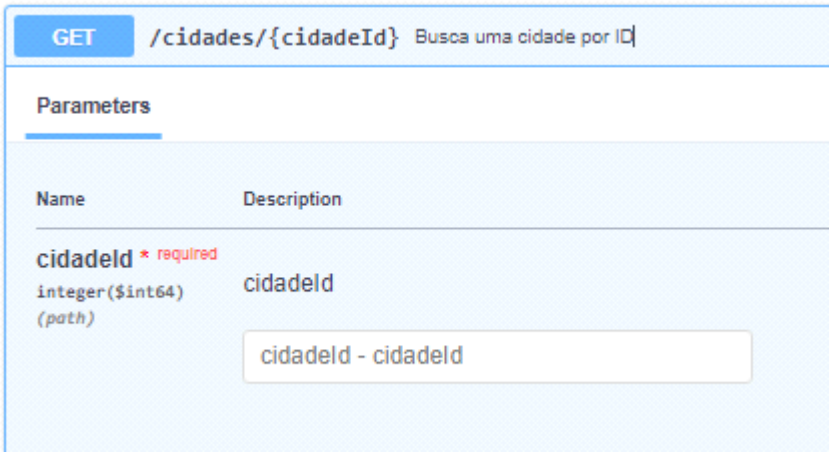
Página do Swagger UI



18.9. Descrevendo parâmetros de entrada na documentação

domingo, 9 de abril de 2023 21:41

- Um parâmetro de entrada é o valor de um path esperado da URI para interagir com o endpoint, ou seja, pode ser um valor de id esperado para interagir com endpoint de busca de cidades por id

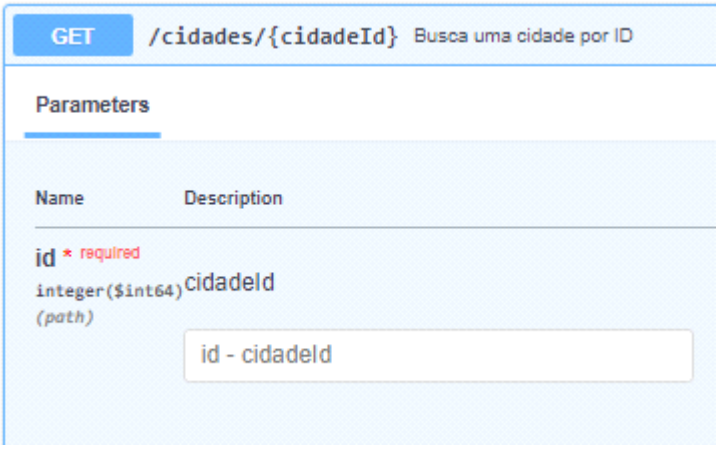


Além disso, podemos customizar a descrição o parâmetro de entrada para tornar a documentação da api mais legível.

Para customizar o Nome do parâmetro, podemos adicionar a anotação @ApiParam na assinatura do método, no campo de parâmetro do tipo.

```
@ApiOperation("Busca uma cidade por ID")
@GetMapping("/{cidadeId}")
public CidadeDTO buscar(@ApiParam(name = "id") @PathVariable Long cidadeId) {
    return cAssembler.toDTO(cidadeService.buscar(cidadeId));
}

@ApiOperation("Atualiza uma cidade por ID")
```

parâmetro

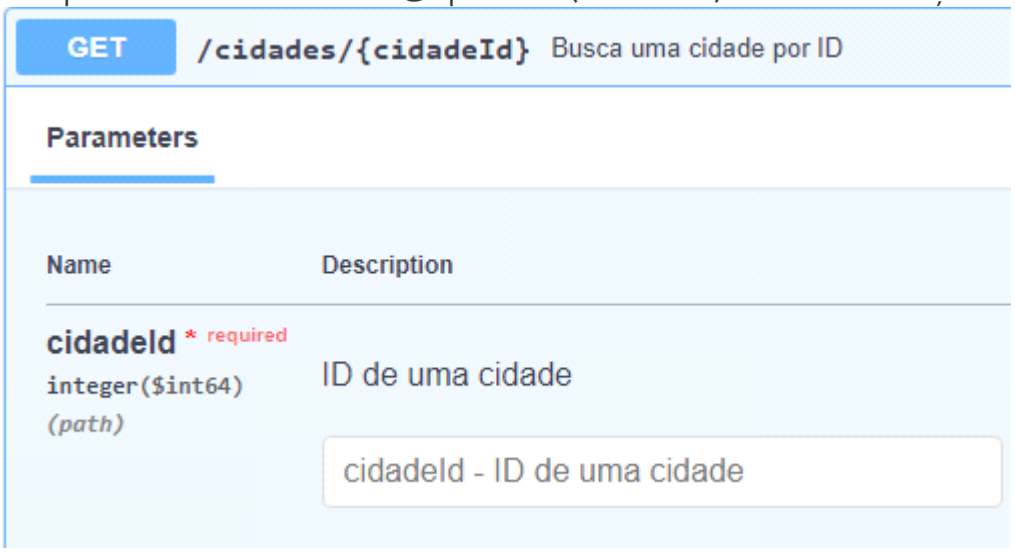


Porém, é interessante deixar por estar auto-explicativo, e podemos alterar apenas a

descrição

```java
@ApiOperation("Busca uma cidade por ID")
@GetMapping("/{cidadeId}")
public CidadeDTO buscar(@ApiParam("ID de uma cidade") @PathVariable Long cidadeId) {
    return cAssembler.toDTO(cidadeService.buscar(cidadeId));
}
```

podemos omitir o parâmetro "value" por default. Logo, @ApiParam(name = "") altera o Name do parâmetro de entrada e @ApiParam(value = "") altera a descrição




```java
@ApiOperation("Cadastra uma cidade")
@PostMapping
public ResponseEntity<CidadeDTO> adicionar(@ApiParam(name = "corpo", value = "Representação de uma nova cidade")
                                           @RequestBody @Valid CidadeInputDTO cidadeInputDTO) {
    try {...} catch (EstadoNaoEncontradoException e) {...}
}
```

No SpringFox 3 e Swagger UI 3 não apresenta descrição do body

- Para retorno de uma Response Entity com Wildcard

```java
@GetMapping
@ApiOperation(value = "Lista todas as cidades cadastradas",
                response = CidadeOutput.class,
                responseContainer="List")
public ResponseEntity<?> listar(){
        List<Cidade> cidades = cidadeRepository.findAll();
        List<CidadeOutput> cidadesOut = cidadeConverter.toCidadeOutList(cidades);
        return ResponseEntity.status(HttpStatus.OK).body(cidadesOut);
}
```

18.10. Descrevendo modelos de representações e suas propriedades

segunda-feira, 10 de abril de 2023 08:20

- Utilizando @ApiModel na entidade que deseja alterar


```java
import ...

9 usages
@ApiModel(value = "cidade", description = "Representa uma cidade")
@Getter
@Setter
public class CidadeDTO {
    private Long id;
    private String nome;
    private EstadoDTO estado;
}
```

- Para descrever as propriedades


```java
@ApiModel(value = "cidade", description = "Representa uma cidade")
@Getter
@Setter
public class CidadeDTO {

    @ApiModelProperty(value = "ID da cidade", example = "1")
    private Long id;

    @ApiModelProperty(example = "Uberlândia")
    private String nome;
    private EstadoDTO estado;
}
```


```java
@Getter
@Setter
public class EstadoDTO {

    @ApiModelProperty(example = "1")
    private Long id;

    @ApiModelProperty(example = "Minas Gerais")
    private String nome;
}
```

18.11. Descrevendo restrições de validação de propriedades do modelo

segunda-feira, 10 de abril de 2023 09:06

- Utilizando @ApiModelProperty com o parâmetro required true ou aproveitar as anotações do Bean validation

```
<dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-bean-validators</artifactId>
        <version>${springfox.version}</version>
    </dependency>
```

```
package com.algaworks.algafood.core.openapi;

import ...

@Configuration
@Import(BeanValidatorPluginsConfiguration.class)
public class SpringFoxConfig {
```

- Não há todas as anotações do Bean Validation implementadas

```
PedidoInputDTO {
  enderecoEntrega*  EnderecoInputDTO {
                      bairro*  string
                      cep*  string
                      cidade*  CidadeRefDTO {
                                 id*  integer($int64)
                               }
                      complemento  string
                      logradouro*  string
                      numero*  string
                    }
  formaPagamento*  FormaPagamentoIdDTO {
                     id*  integer($int64)
                   }
  itensPedido*  [ItemPedidoInputDTO {
                  observacao  string
                  produtoId*  integer($int64)
                  quantidade*  integer($int32)
                }]
  restaurante*  RestauranteIdDTO {
                  id*  integer($int64)
                }
}
```

Para reparar alguns bugs, é necessário anotar com um exemplo com @ApiModelProperty e passar o parâmetro required

```
Schemas

Cidade >

CidadeInputDTO {
  estado*  EstadoIdInputDTO {
             id  integer($int64)
           }
  nome*  string
         example: Uberlândia
}
```

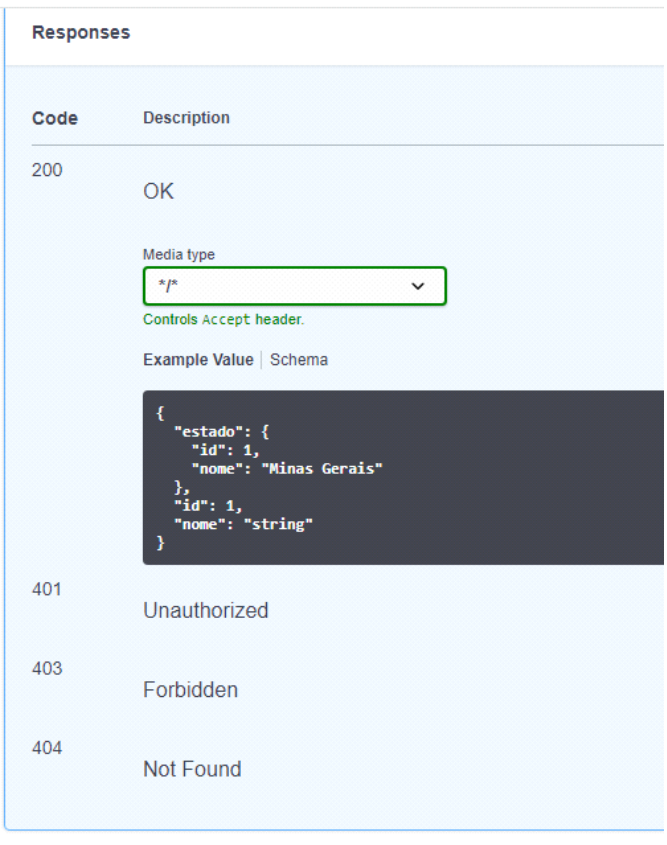
```
6 usages
@Getter
@Setter
public class CidadeInputDTO {

    @ApiModelProperty(example = "Uberlândia", required = true)
    @NotBlank(message = "Não é permitido um campo vazio")
    private String nome;

    @ApiModelProperty(example = "1", required = true)
    @Valid
    @NotNull
    private EstadoIdInputDTO estado;
}
```

```
@Getter
@Setter
public class EstadoIdInputDTO {

    @ApiModelProperty(required = true)
    @NotNull
    private Long id;
}
```

```
CidadeInputDTO {
  estado*  EstadoIdInputDTO {
             id*  integer($int64)
           }
  nome*  string
         example: Uberlândia
}
```

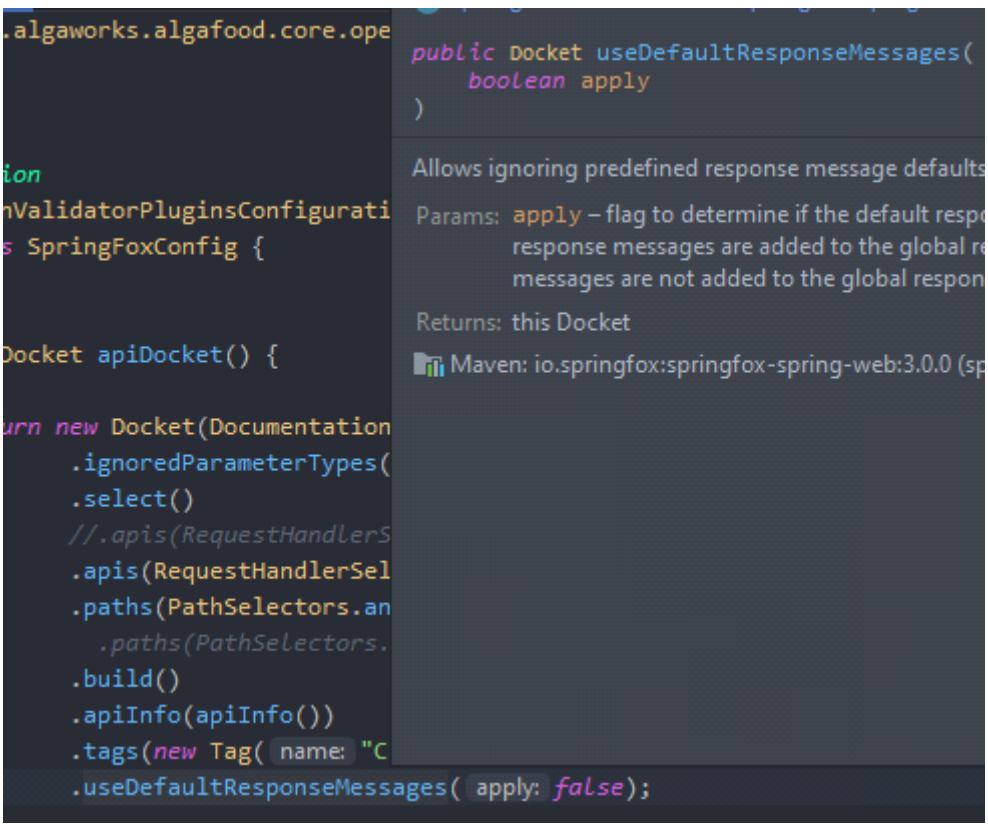
18.12. Descrevendo códigos de status de respostas de forma global

segunda-feira, 10 de abril de 2023 10:56



- O Spring Fox consegue distinguir as respostas de sucesso por conta das anotações do Spring e o Response Entity Retornado, mas as respostas de erro não, entretanto, ele deduz os possíveis status . Podemos alterar as informações sobre o status de erro

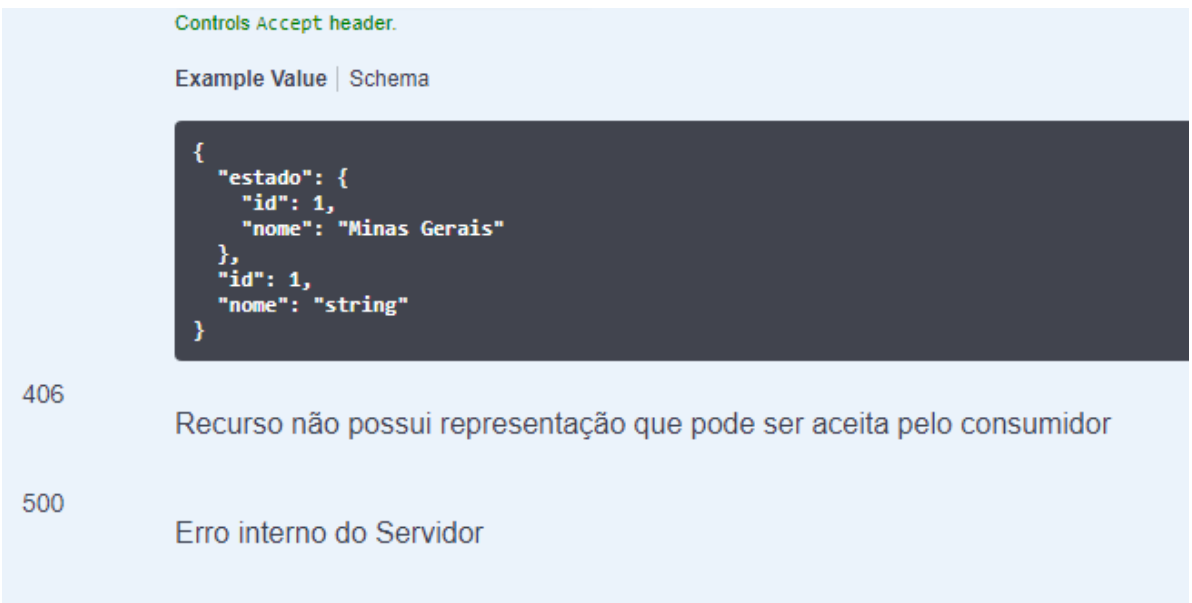
- Desabilitar status de possíveis erro com useDefaultResponseMessages();



- Tratar de forma global o status de erro dos endpoints na documentação

```
.tags(new Tag( name: "Cidades",  description: "Gerencia as cidades"))
.useDefaultResponseMessages( apply: false)
.globalResponses(HttpMethod.GET, globalResponses());
```

```
private List<Response> globalResponses() {
    return Arrays.asList(
            new ResponseBuilder()
                    .code(String.valueOf(HttpStatus.INTERNAL_SERVER_ERROR.value()))
                    .description("Erro interno do Servidor")
                    .build(),
            new ResponseBuilder()
                    .code(String.valueOf(HttpStatus.NOT_ACCEPTABLE.value()))
                    .description("Recurso não possui representação que pode ser aceita pelo consumidor")
                    .build()
    );
}
```



- O Spring Fox prioriza as mensagens a nível de método, por exemplo, status de sucesso que são declarados explicitamente nos cabeçalhos e retornos dos métodos, como 200 ou 204. Para alterar isso, devem ser feitos de forma específica e não global

18.13. Desafio: descrevendo códigos de status de respostas de forma global

segunda-feira, 10 de abril de 2023 15:14

18.14. Descrevendo o modelo de representação de problema

segunda-feira, 10 de abril de 2023 15:24

- Na aula 8.17. Conhecendo a RFC 7807 (Problem Details for HTTP APIs) adicionamos um modelo de representação de erros, mas o Spring Fox não mapeou na documentação, podemos utilizar o mesmo modelo na documentação para representar os erros de requisição de endpoint

- Adicionar em Schemas, o modelo de representação de problemas de acordo com a RFC 7807
  - Adicionar instância de TypeResolve

```
@Configuration
@Import(BeanValidatorPluginsConfiguration.class)
public class SpringFoxConfig {

    1 usage
    TypeResolver typeResolver = new TypeResolver();
```

  - Adicionar no Builder additionalModels

```
.useDefaultResponseMessages( apply: false)
.globalResponses(HttpMethod.GET, globalGetResponses())
.globalResponses(HttpMethod.POST, globalPutResponses())
.globalResponses(HttpMethod.PUT, globalPutResponses())
.globalResponses(HttpMethod.DELETE, globalDeleteResponses())
.additionalModels(typeResolver.resolve(Problem.class));
```

```java
/*Classe criada para representar a mensagem d
17 usages
@ApiModel("Problema")
@Getter
@Builder
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Problem {

    private Integer status;
    private String type;
    private String title;
    private String detail;
    /*Extensão da especificação*/
    private String userMessage;
    private OffsetDateTime timeStamp;
    /*Extensão cap 9 - aula 4 */
    private List<Object> objects;

    @ApiModel("ObjetoProblema")
    @Getter
    @Builder
    public static class Object {
        private String name;
        private String userMessage;
    }
}
```

- Descrever os exemplos

```java
@ApiModelProperty(example = "400", position = 1)
private Integer status;

@ApiModelProperty(example = "2019-12-01T18:09:02.70844Z", position = 5)
private OffsetDateTime timeStamp;

@ApiModelProperty(example = "https://algafood.com.br/dados-invalidos", position = 10)
private String type;

@ApiModelProperty(example = "Dados inválidos", position = 15)
private String title;

@ApiModelProperty(example = "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.",
        position = 20)
private String detail;

/*Extensão da especificação*/
@ApiModelProperty(example = "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.",
        position = 25)
private String userMessage;
```

```java
@ApiModelProperty(value = "Lista de objetos ou campos que geraram o erro (opcional)",
        position = 30)
/*Extensão cap 9 - aula 4 */
private List<Object> objects;

@ApiModel("ObjetoProblema")
@Getter
@Builder
public static class Object {

    @ApiModelProperty(example = "preco")
    private String name;

    @ApiModelProperty(example = "O preço é obrigatório")
    private String userMessage;
}
```

Erro:
O SpringFox possui um Serializer do Jackson que é configurado internamente, que não carrega as configurações do SpringBoot. Ao tentar serializar um OffsetDateTime em algum example da nossa documentação, nos depararemos com a seguinte Exception:

```
java.lang.RuntimeException Create breakpoint : Could not write JSON
    at springfox.documentation.spring.web.json.JsonSerializer.toJson(JsonSerializer.java:40)
    at springfox.documentation.oas.web.OpenApiControllerWebMvc.getDocumentation(OpenApiControllerWebMvc.java:95)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:670) <1 internal line>
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:779) <37 internal lines>
Caused by: com.fasterxml.jackson.databind.exc.InvalidDefinitionException Create breakpoint : Java 8 date/time type
```

Para resolver o problema, primeiramente é necessário adicionar uma dependência no pom.xml

```
<dependency>
<groupId>com.fasterxml.jackson.datatype</groupId>
<artifactId>jackson-datatype-jsr310</artifactId>
</dependency>
```

Em seguida crie um Bean do tipo JacksonModuleRegistrar na sua classe SpringFoxConfig, para fazer com que o SpringFox carregue o módulo de conversão de datas:

```java
@Bean
public JacksonModuleRegistrar springFoxJacksonConfig() {
    return objectMapper -> objectMapper.registerModule(new JavaTimeModule());
}
```

Problema {

detail	string *example: Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.*
objects	[Lista de objetos ou campos que geraram o erro (opcional)

ObjetoProblema {

name	string *example: preco*
userMessage	string *example: O preço é obrigatório*

}]

status	integer($int32) *example: 400*
timeStamp	string($date-time) *example: 2019-12-01T18:09:02.70844Z*
title	string *example: Dados inválidos*
type	string *example: https://algafood.com.br/dados-invalidos*
userMessage	string *example: Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.*

}

18.15. Referenciando modelo de representação de problema com códigos de status de erro

segunda-feira, 10 de abril de 2023 17:16

Baseado na aula 18.14. Descrevendo o modelo de representação de problema podemos adicionar o modelo de representação de problema na documentação quando há possibilidade de erro de requisição

Configurando o model de erro Global
O método responseModel não existe na classe ResponseBuilder do SpringFox 3, assim como o ModelRef que foi depreciado. Vamos utilizar os métodos representation e apply para realizar a mesma configuração.

Primeiramente na classe SpringFoxConfig precisamos criar um método que ira gerar a referência para classe Problem:

```
private Consumer<RepresentationBuilder> getProblemaModelReference() {
  return r -> r.model(m -> m.name("Problema")
      .referenceModel(ref -> ref.key(k -> k.qualifiedModelName(
          q ->
q.name("Problema").namespace("com.algaworks.algafood.api.exceptionhandler")))))
;
}
```

Em seguida aplicar essa configuração nas respostas globais, além de utilizar o método representation com o parametro MediaType.APPLICATION_JSON:

```
private List<Response> globalGetResponseMessages() {
  return Arrays.asList(
      new ResponseBuilder()
          .code(String.valueOf(HttpStatus.INTERNAL_SERVER_ERROR.value()))
          .description("Erro interno do Servidor")
          .representation( MediaType.APPLICATION_JSON )
          .apply(getProblemaModelReference())
          .build(),
      new ResponseBuilder()
          .code(String.valueOf(HttpStatus.NOT_ACCEPTABLE.value()))
          .description("Recurso não possui representação que pode ser aceita pelo
consumidor")
          .build()
  );
}

private List<Response> globalPostPutResponseMessages() {
  return Arrays.asList(
      new ResponseBuilder()
          .code(String.valueOf(HttpStatus.BAD_REQUEST.value()))
          .description("Requisição inválida (erro do cliente)")
```

```
            .representation( MediaType.APPLICATION_JSON )
            .apply(getProblemaModelReference())
            .build(),
        new ResponseBuilder()
            .code(String.valueOf(HttpStatus.INTERNAL_SERVER_ERROR.value()))
            .description("Erro interno no servidor")
            .representation( MediaType.APPLICATION_JSON )
            .apply(getProblemaModelReference())
            .build(),
        new ResponseBuilder()
            .code(String.valueOf(HttpStatus.NOT_ACCEPTABLE.value()))
            .description("Recurso não possui representação que poderia ser aceita pelo
consumidor")
            .build(),
        new ResponseBuilder()
            .code(String.valueOf(HttpStatus.UNSUPPORTED_MEDIA_TYPE.value()))
            .description("Requisição recusada porque o corpo está em um formato não
suportado")
            .representation( MediaType.APPLICATION_JSON )
            .apply(getProblemaModelReference())
            .build()
    );
}

private List<Response> globalDeleteResponseMessages() {
    return Arrays.asList(
        new ResponseBuilder()
            .code(String.valueOf(HttpStatus.BAD_REQUEST.value()))
            .description("Requisição inválida (erro do cliente)")
            .representation( MediaType.APPLICATION_JSON )
            .apply(getProblemaModelReference())
            .build(),
        new ResponseBuilder()
            .code(String.valueOf(HttpStatus.INTERNAL_SERVER_ERROR.value()))
            .description("Erro interno no servidor")
            .representation( MediaType.APPLICATION_JSON )
            .apply(getProblemaModelReference())
            .build()
    );
}
```

400

Requisição inválida (erro do cliente)

Media type

application/json

Example Value | Schema


```
{
  "detail": "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.",
  "objects": [
    {
      "name": "preco",
      "userMessage": "O preço é obrigatório"
    }
  ],
  "status": 400,
  "timeStamp": "2019-12-01T18:09:02.70844Z",
  "title": "Dados inválidos",
  "type": "https://algafood.com.br/dados-invalidos",
  "userMessage": "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente."
}
```

18.16. Descrevendo códigos de status de respostas em endpoints específicos

segunda-feira, 10 de abril de 2023 17:30

Para documentar status de erros de endpoints específicos, podemos tratar individualmente cada método, em vez de tratar globalmente todas as requisições HTTP

Uma cidade pode retornar um status 404 not found ao passar um id inexistente, que retorna o erro de recurso não encontrado. Ou um 400 bad request ao passar um tipo de parâmetro de url inválido.

Utilizando a anotação @ApiResponses que recebe um array de @ApiResponse (io.swagger.v3.oas.annotations.responses)

```java
@ApiOperation("Busca uma cidade por ID")
@ApiResponses({
        @ApiResponse(responseCode = "400", description = "ID da cidade inválido", content = @Content(schema = @Schema(implementation = Problem.class))),
        @ApiResponse(responseCode = "404", description = "Cidade não encontrada", content = @Content(schema = @Schema(implementation = Problem.class)))
})
@GetMapping("/{cidadeId}")
public CidadeDTO buscar(@ApiParam("ID de uma cidade") @PathVariable Long cidadeId) {
    return cAssembler.toDTO(cidadeService.buscar(cidadeId));
}
```

400

Requisição inválida (erro do cliente)

Media type

application/json

Example Value | Schema

```
{
  "detail": "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente.",
  "objects": [
    {
      "name": "preco",
      "userMessage": "O preço é obrigatório"
    }
  ],
  "status": 400,
  "timeStamp": "2019-12-01T18:09:02.70844Z",
  "title": "Dados inválidos",
  "type": "https://algafood.com.br/dados-invalidos",
  "userMessage": "Um ou mais campos estão inválidos. Faça o preenchimento correto e tente novamente."
}
```

Também podemos customizar outras mensagens de status

Responses

Code	Description
204	No Content
400	Requisição inválida (erro do cliente) Media type: application/json Example Value | Schema

```
{
  "detail": "Um ou mais campos estão inválidos. Faça o pr
  "objects": [
    {
      "name": "preco",
      "userMessage": "O preço é obrigatório"
    }
  ],
  "status": 400,
  "timeStamp": "2019-12-01T18:09:02.70844Z",
  "title": "Dados inválidos",
  "type": "https://algafood.com.br/dados-invalidos",
  "userMessage": "Um ou mais campos estão inválidos. Faça
}
```

500	Erro interno no servidor

Media type

Por padrão, no SpringFox 3 e Swagger UI 3, o status e descrição vem 200 OK, mesmo em endpoints retornando CREATED. Podemos alterar

```java
@ApiOperation("Cadastra uma cidade")
@PostMapping
@ApiResponses(@ApiResponse(responseCode = "201", description = "Cidade cadastrada"))
public ResponseEntity<CidadeDTO> adicionar(@ApiParam(name = "corpo", value = "Representação de uma nova cidade")
                                           @RequestBody @Valid CidadeInputDTO cidadeInputDTO) {
    try {
        final Cidade cidade = cidadeInputDisassembler.DTOtoDomainModel(cidadeInputDTO);
        Cidade c = cidadeService.salvar(cidade);
        return ResponseEntity.status(HttpStatus.CREATED).body(cAssembler.toDTO(c));
    } catch (EstadoNaoEncontradoException e) {
        throw new NegocioException(e.getMessage(), e);
    }
}
```

Code	Description
200	OK Media type: */* Controls Accept header. Example Value | Schema

```
{
  "estado": {
    "id": 1,
    "nome": "Minas Gerais"
  },
  "id": 1,
  "nome": "string"
}
```

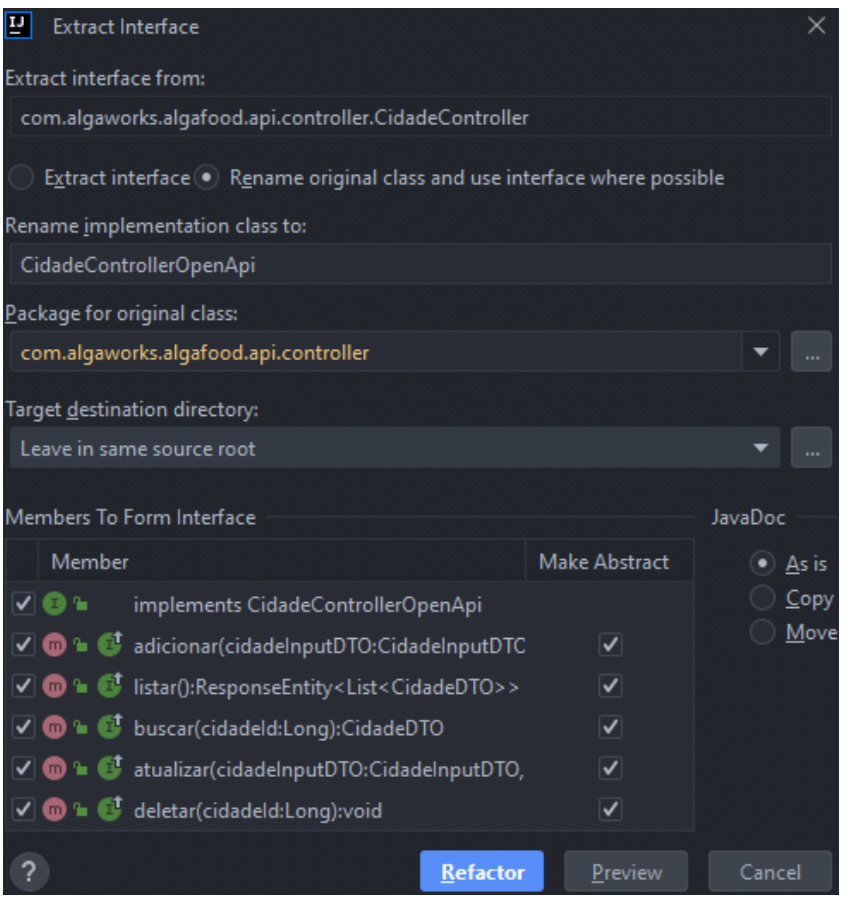
201	Cidade cadastrada
400	Requisição inválida (erro do cliente)

Media type

18.17. Desacoplando anotações do Swagger dos controladores

segunda-feira, 10 de abril de 2023 18:09

- Podemos criar uma interface da classe controladora e anotar os métodos da interface nova com anotações do SpringFox 3




```
2 usages  1 implementation
public interface CidadeControllerOpenApi {

    1 implementation
    @ApiOperation("Cadastra uma cidade")
    @PostMapping
    @ApiResponse(responseCode = "201", description = "Cidade cadastrada")
    ResponseEntity<CidadeDTO> adicionar(@ApiParam(name = "corpo", value = "Representação de uma nova cidade")
                                        @RequestBody @Valid CidadeInputDTO cidadeInputDTO);

    1 implementation
    @ApiOperation("Lista as cidades")
    @GetMapping
    ResponseEntity<List<CidadeDTO>> listar();

    1 implementation
    @ApiOperation("Busca uma cidade por ID")
    @ApiResponses({
```

- Não muda nada na documentação, porém, deixa a classe controladora menos poluída

18.18. Desafio: descrevendo documentação de endpoints de grupos

segunda-feira, 10 de abril de 2023 18:21

1 - Alterando SpringFoxConfig
Primeiro, vamos alterar o método apiDocket, adicionando novas Tags

```
.tags(new Tag("Cidades", "Gerencia as cidades"),
    new Tag("Grupos", "Gerencia os grupos de usuários"));
```

2 - Alterando GrupoInput
Vamos adicionar a anotação @ApiModelProperty acima da propriedade "nome"

```
@ApiModelProperty(example = "Gerente", required = true)
@NotBlank
private String nome;
```

3 - Alterando GrupoModel
Agora, vamos adicionar a mesma anotação nas propriedades de nosso modelo

```
@ApiModelProperty(example = "1")
private Long id;

@ApiModelProperty(example = "Gerente")
private String nome;
```

4 - Criando GrupoControllerOpenApi

```
@Api(tags = "Grupos")
public interface GrupoControllerOpenApi {

  @ApiOperation("Lista os grupos")
  public List<GrupoModel> listar();

  @ApiOperation("Busca um grupo por ID")
  @ApiResponses({
    @ApiResponse(code = 400, message = "ID da grupo inválido", response =
Problem.class),
    @ApiResponse(code = 404, message = "Grupo não encontrado", response =
Problem.class)
  })
  public GrupoModel buscar(
      @ApiParam(value = "ID de um grupo", example = "1")
      Long grupoId);

  @ApiOperation("Cadastra um grupo")
  @ApiResponses({
    @ApiResponse(code = 201, message = "Grupo cadastrado"),
  })
  public GrupoModel adicionar(
      @ApiParam(name = "corpo", value = "Representação de um novo grupo")
      GrupoInput grupoInput);

  @ApiOperation("Atualiza um grupo por ID")
  @ApiResponses({
    @ApiResponse(code = 200, message = "Grupo atualizado"),
    @ApiResponse(code = 404, message = "Grupo não encontrado", response =
Problem.class)
  })
  public GrupoModel atualizar(
      @ApiParam(value = "ID de um grupo", example = "1")
      Long grupoId,

      @ApiParam(name = "corpo", value = "Representação de um grupo com os novos
dados")
      GrupoInput grupoInput);

  @ApiOperation("Exclui um grupo por ID")
  @ApiResponses({
    @ApiResponse(code = 204, message = "Grupo excluído"),
    @ApiResponse(code = 404, message = "Grupo não encontrado", response =
Problem.class)
  })
  public void remover(
      @ApiParam(value = "ID de um grupo", example = "1")
      Long grupoId);

}
```

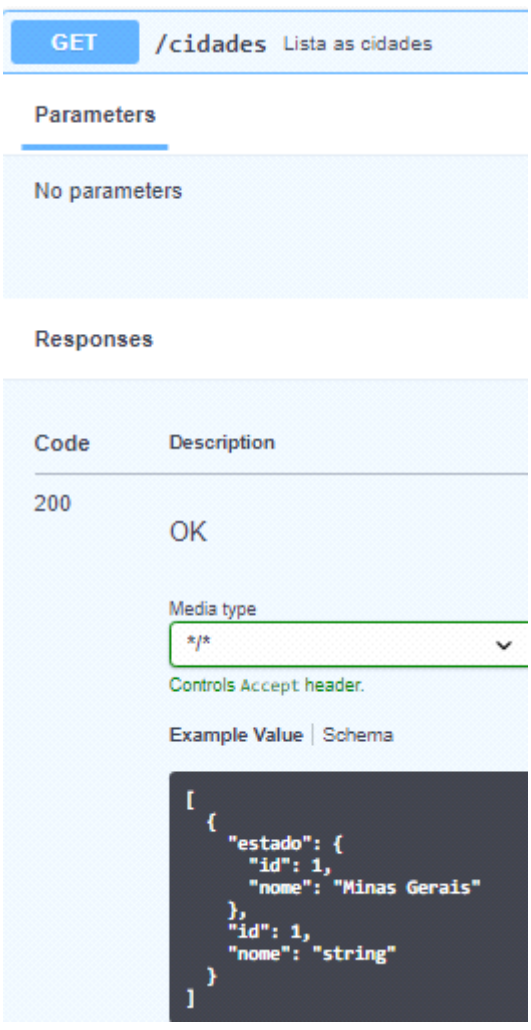
5 - Alterando GrupoController
Vamos alterar GrupoController para que o mesmo implemente a interface GrupoControllerOpenApi

```
public class GrupoController implements GrupoControllerOpenApi {
  //...
}
```

18.19. Descrevendo media type da resposta nos endpoints

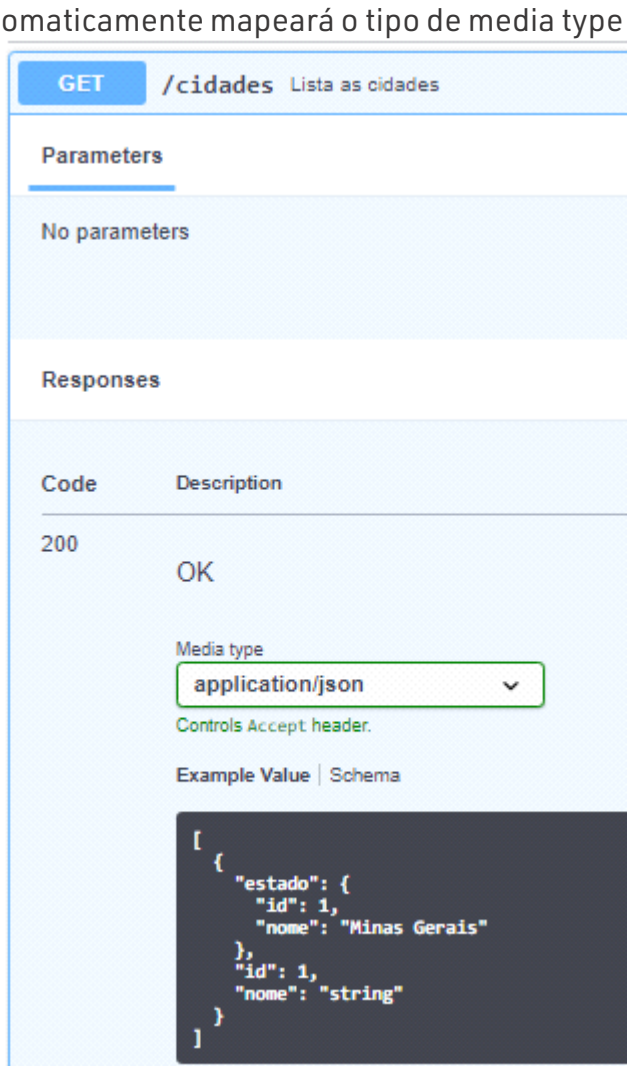
segunda-feira, 10 de abril de 2023 18:52

- Na documentação, um endpoint é descrito que aceita qualquer tipo de media type, mas podemos ser mais específicos, declarando o tipo de media type aceito no controlador




```
package com.algaworks.algafood.api.controller;

import ...

@stController
@RequestMapping(path = "/grupos", produces = MediaType.APPLICATION_JSON_VALUE)
public class GrupoController implements com.algaworks.algafood.api.controller.openapi.GrupoControllerOpenApi {

    6 usages
```

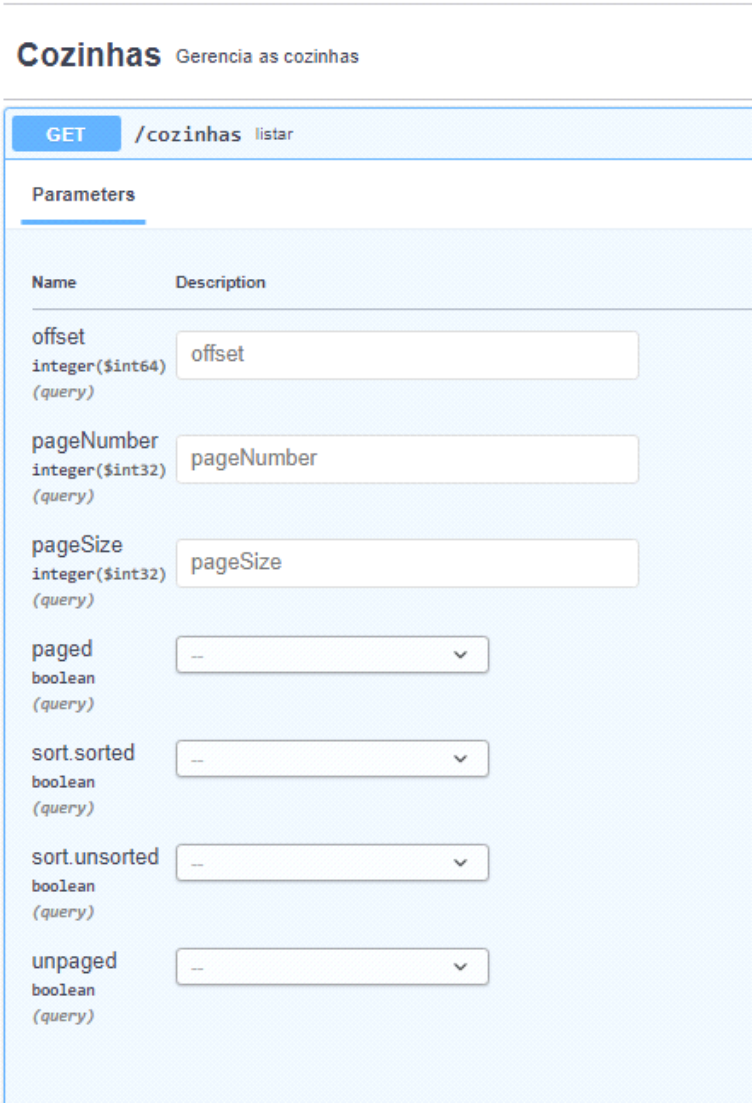
O SpringFox automaticamente mapeará o tipo de media type aceita

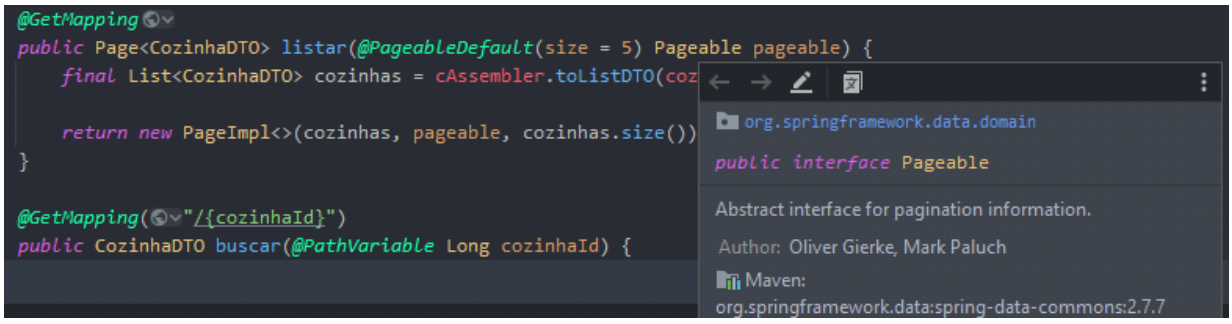


18.20. Corrigindo documentação com substituição de Pageable

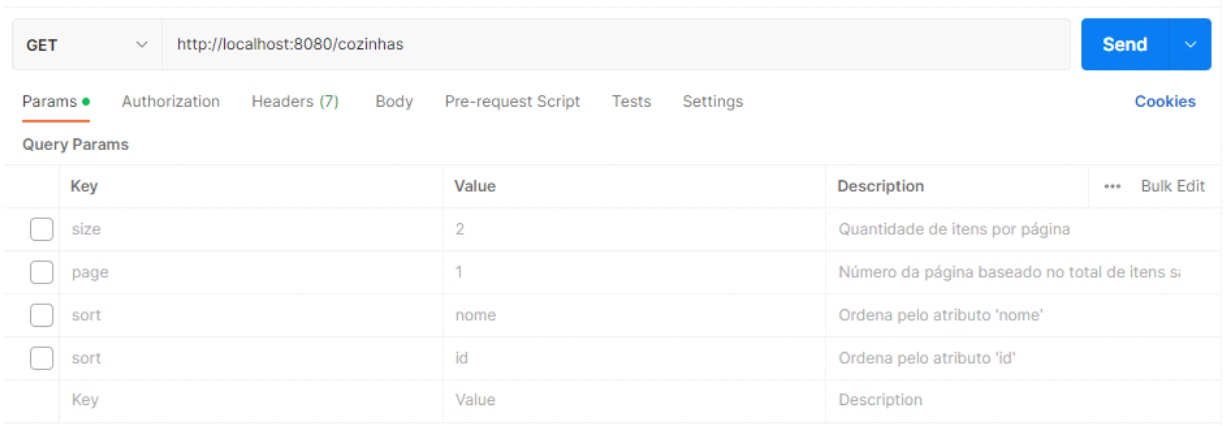
segunda-feira, 10 de abril de 2023 21:10

- Em requisições paginadas, o Swagger UI acaba mapeando parâmetros da interface Pageable, que implementa endpoints para paginação





para contornar isso, podemos criar uma entidade que representa as propriedades que precisamos da interface Pageable (sort, page e size)

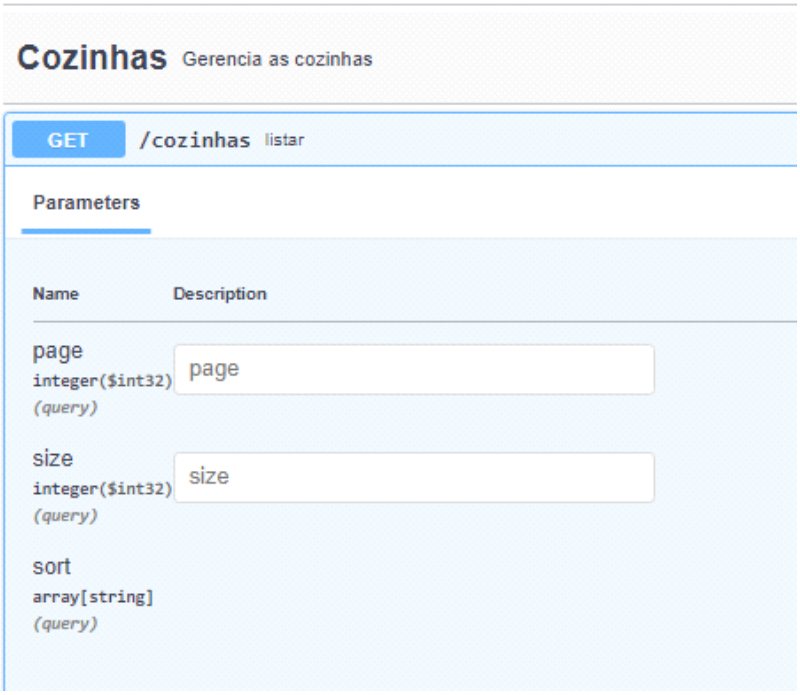


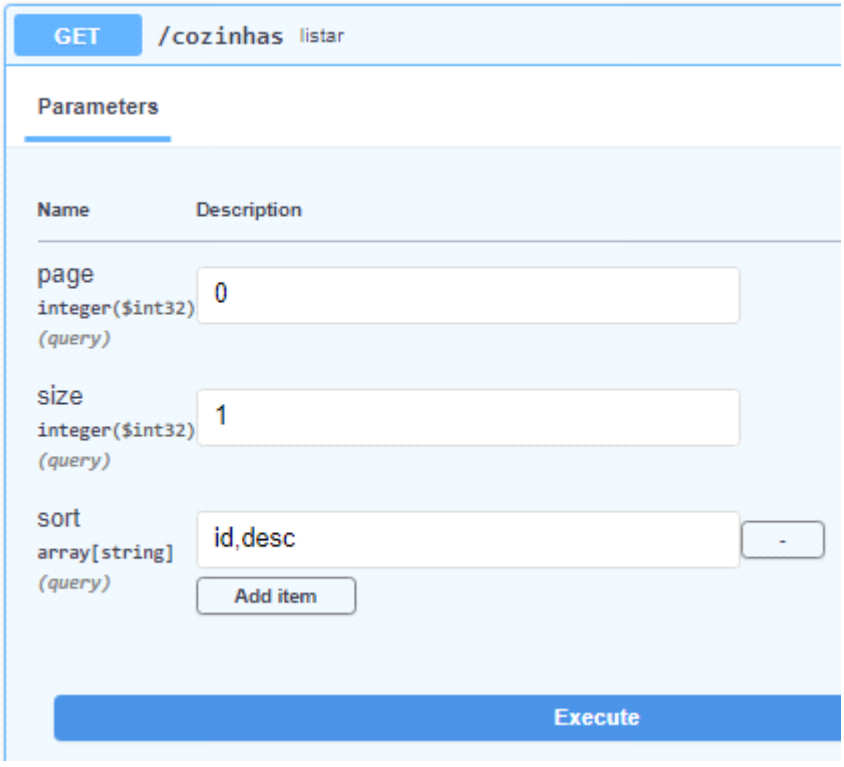
A classe será criada para fins de documentação, como se fosse um DTO para a documentação do Swagger UI

```java
package com.algaworks.algafood.core.openapi.model;

import ...

2 usages
@Setter
@Getter
public class PageableModelOpenApi {

    private int page;
    private int size;
    private List<String> sort; /*Podemos ter mais de um sort no query params da requisição*/
}
```

```java
@Bean
public Docket apiDocket() {

    return new Docket(DocumentationType.OAS_30)
            .ignoredParameterTypes(ignoredParameterTypes())
            .select()
            //.apis(RequestHandlerSelectors.any())/*Todos os endpointds*/
            .apis(RequestHandlerSelectors.basePackage("com.algaworks.algafood.api"))
            .paths(PathSelectors.any())/*Caminho padrão*/
              .paths(PathSelectors.ant("/restaurantes/*"))
            .build()
            .apiInfo(apiInfo())
            .tags(tags()[0], tags())
            .useDefaultResponseMessages( apply: false)
            .globalResponses(HttpMethod.GET, globalGetResponses())
            .globalResponses(HttpMethod.POST, globalPutResponses())
            .globalResponses(HttpMethod.PUT, globalPutResponses())
            .globalResponses(HttpMethod.DELETE, globalDeleteResponses())
            .additionalModels(typeResolver.resolve(Problem.class))
            .directModelSubstitute(Pageable.class, PageableModelOpenApi.class);
}
```

Swagger UI


Cozinhas Gerencia as cozinhas
GET /cozinhas listar
Parameters
Name Description
page
integer($int32)
(query)
page
size
integer($int32)
(query)
size
sort
array[string]
(query)


GET /cozinhas listar
Parameters
Name Description
page
integer($int32)
(query)
0
size
integer($int32)
(query)
1
sort
array[string]
(query)
id,desc
-
Add item
Execute

Responses

Curl

```
curl -X GET "http://localhost:8080/cozinhas?page=0&size=1&sort=id%2Cdesc" -H "accept: */*"
```

Request URL

```
http://localhost:8080/cozinhas?page=0&size=1&sort=id%2Cdesc
```

Server response

Code	Details
200	Response body

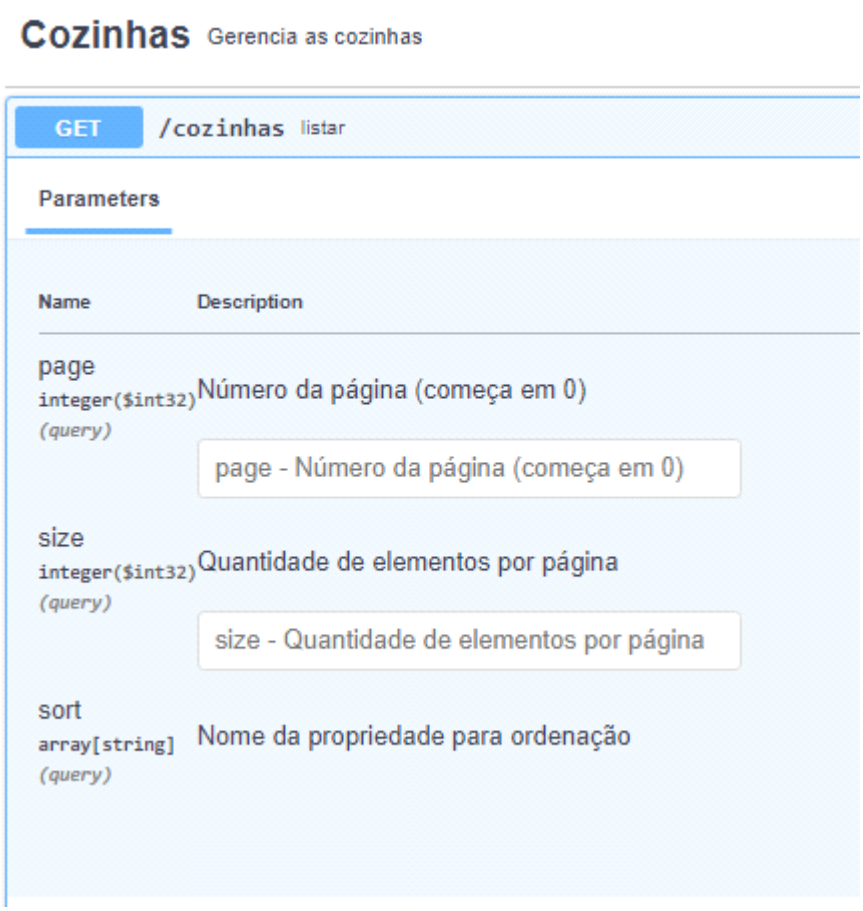
```
{
  "content": [
    {
      "id": 4,
      "nome": "Brasileira"
    }
  ],
  "size": 1,
  "totalElements": 1,
  "totalPages": 1,
  "number": 0
}
```

Response headers

```
connection: keep-alive
content-length: 95
content-type: application/json
date: Tue11 Apr 2023 00:23:41 GMT
etag: "051dc5657702dcfc278ba4cdf19748e23"
keep-alive: timeout=60
vary: OriginAccess-Control-Request-MethodAccess-Control-Request-Headers
```

Podemos ainda tratar as descrições

```java
import ...
2 usages
@ApiModel("Pageable")
@Setter
@Getter
public class PageableModelOpenApi {

    @ApiModelProperty(example = "0", value = "Número da página (começa em 0)")
    private int page;

    @ApiModelProperty(example = "10", value = "Quantidade de elementos por página")
    private int size;

    @ApiModelProperty(example = "nome,asc", value = "Nome da propriedade para ordenação")
    private List<String> sort; /*Podemos ter mais de um sort no query params da requisição
}
```

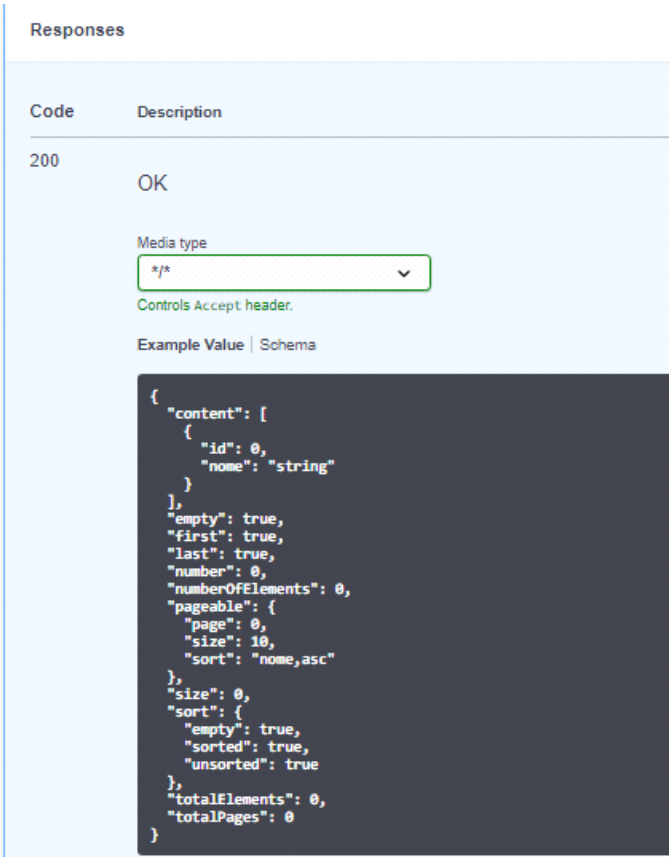

Cozinhas
Gerencia as cozinhas
GET
/cozinhas
listar
Parameters
Name
Description
page
integer($int32)
(query)
Número da página (começa em 0)
page - Número da página (começa em 0)
size
integer($int32)
(query)
Quantidade de elementos por página
size - Quantidade de elementos por página
sort
array[string]
(query)
Nome da propriedade para ordenação

18.21. Corrigindo documentação com substituição de Page

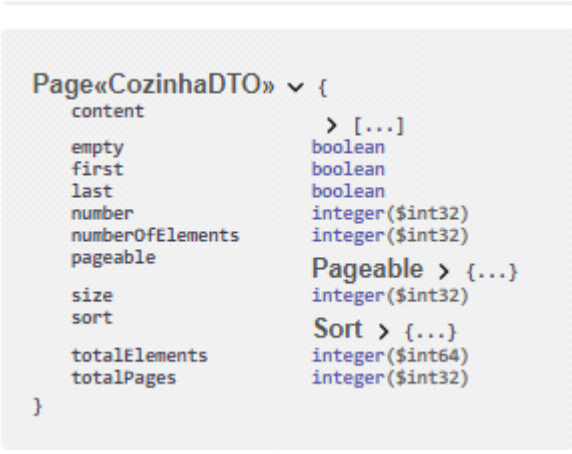
segunda-feira, 10 de abril de 2023 21:34

Na resposta, o Swagger UI mapeia todas as informações da entidade paginada, já que na aula 13.10. Implementando JsonSerializer para customizar representação de paginação foi utilizado uma classe para representar as informações da paginação. A ideia é deixar as informações da paginação da api o mais fiel possível na documentação

- Resposta da requisição no swagger UI



- Schema no Swagger UI

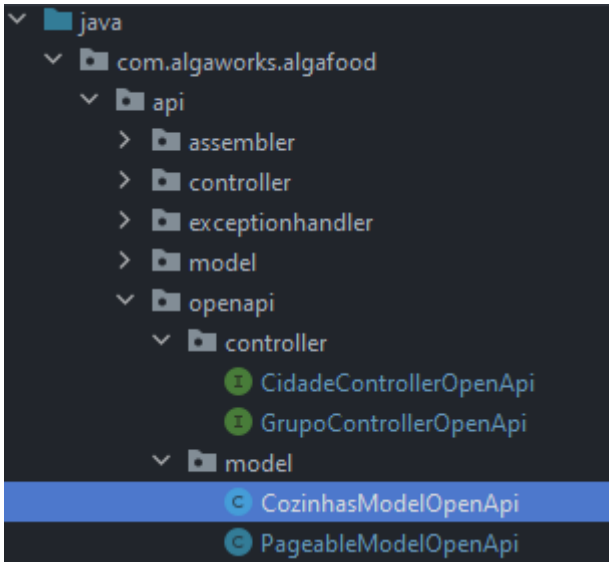


- Controlador

```java
@GetMapping
public Page<CozinhaDTO> listar(@PageableDefault(size = 5) Pageable pageable) {
    final List<CozinhaDTO> cozinhas = cAssembler.toListDTO(cozinhaService.listar(pageable).getContent());

    return new PageImpl<>(cozinhas, pageable, cozinhas.size());
}
```

Temos que fazer o ajuste da documentação, criando uma classe para representar o modelo para representar na documentação.

- Refatoração na estrutura de pacotes



foi criada a classe CozinhasModelOpenApi para substituir a representação da resposta da entidade paginada

```java
package com.algaworks.algafood.api.openapi.model;

import ...

@ApiModel("CozinhasModel")
@Getter
@Setter
public class CozinhasModelOpenApi {

    private List<CozinhaDTO> content;
    private Long size;
    private Long totalElements;
    private Long totalPages;
    private Long number;
}
```

- Diferente do que foi implementado na aula [18.20. Corrigindo documentação com substituição de Pageable] utilizamos directModels

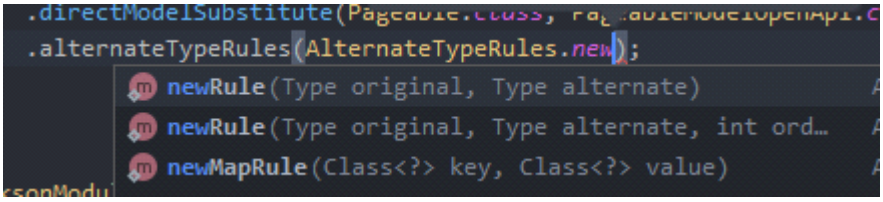
```java
.additionalModels(typeResolver.resolve(Problem.class))
.directModelSubstitute(Pageable.class, PageableModelOpenApi.class);
```

para substituir uma classe por outra, porém, estamos trabalhando com uma classe que usa generics (genérica)

```
@GetMapping
public Page<CozinhaDTO> listar(@Pag
    final List<CozinhaDTO> cozinhas

    return new PageImpl<>(cozinhas,
}
```

um page de cozinhasDto

logo, temos que usar AlternateTypeRules com o argumento AlternateTypeRules.newRule usando o TypeResolve




```
.alternateTypeRules(AlternateTypeRules.newRule(
        typeResolver.resolve(Page.class, CozinhaDTO.class), CozinhasModelOpenApi.class)
);
```

Swagger UI

Responses

Code	Description
200	OK Media type: */* Controls Accept header. Example Value | Schema

```
{
  "content": [
    {
      "id": 0,
      "nome": "string"
    }
  ],
  "number": 0,
  "size": 0,
  "totalElements": 0,
  "totalPages": 0
}
```

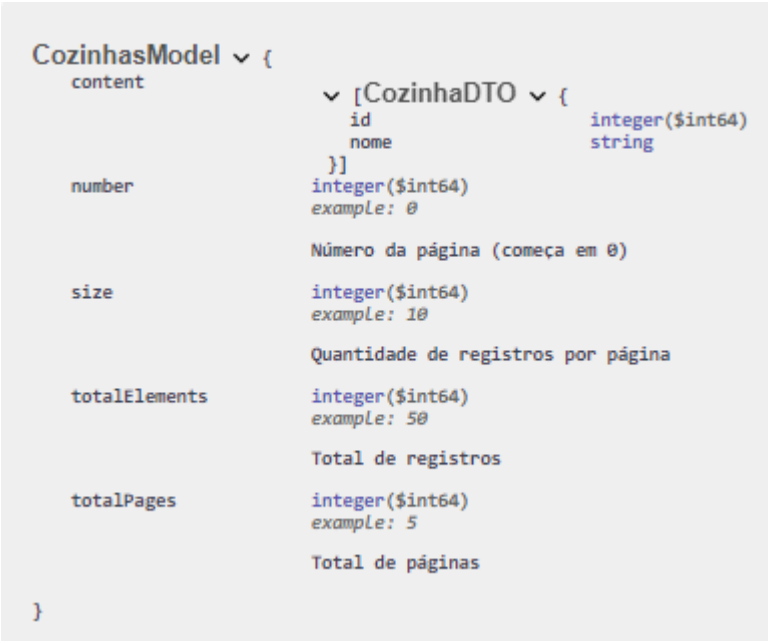
406	Recurso não possui representação que pode ser aceita pelo consumidor

```java
package com.algaworks.algafood.api.openapi.model;

import ...

2 usages
@ApiModel("CozinhasModel")
@Getter
@Setter
public class CozinhasModelOpenApi {

    private List<CozinhaDTO> content;

    @ApiModelProperty(example = "10", value = "Quantidade de registros por página")
    private Long size;

    @ApiModelProperty(example = "50", value = "Total de registros")
    private Long totalElements;

    @ApiModelProperty(example = "5", value = "Total de páginas")
    private Long totalPages;

    @ApiModelProperty(example = "0", value = "Número da página (começa em 0)")
    private Long number;
}
```

- Caso tenha mais entidades que utilizem pesquisa paginada, temos que fazer os mesmos passos para cada classe. Podemos tornar o código mais reutilizável

```java
2 usages
@ApiModel("CozinhasModel")
public class CozinhasModelOpenApi extends PagedModelOpenApi<CozinhaDTO> {

}
```

```java
import java.util.List;

1 usage  1 inheritor
@Getter
@Setter
public class PagedModelOpenApi<T> {
    private List<T> content;

    @ApiModelProperty(example = "10", value = "Quantidade de registros por página")
    private Long size;

    @ApiModelProperty(example = "50", value = "Total de registros")
    private Long totalElements;

    @ApiModelProperty(example = "5", value = "Total de páginas")
    private Long totalPages;

    @ApiModelProperty(example = "0", value = "Número da página (começa em 0)")
    private Long number;
}
```


CozinhasModel {
content
[CozinhaDTO {
id integer($int64)
nome string
}]
number integer($int64)
example: 0
Número da página (começa em 0)
size integer($int64)
example: 10
Quantidade de registros por página
totalElements integer($int64)
example: 50
Total de registros
totalPages integer($int64)
example: 5
Total de páginas
}

18.22. Desafio: descrevendo documentação de endpoints de cozinhas

segunda-feira, 10 de abril de 2023 22:25

1 - Alterando SpringFoxConfig
Primeiro, vamos alterar o método apiDocket, adicionando novas Tags

```
.tags(new Tag("Cidades", "Gerencia as cidades"),
      new Tag("Grupos", "Gerencia os grupos de usuários"),
      new Tag("Cozinhas", "Gerencia as cozinhas"));
```

2 - Alterando CozinhaInput
Vamos adicionar a anotação @ApiModelProperty acima da propriedade "nome"

```
@ApiModelProperty(example = "Brasileira", required = true)
@NotBlank
private String nome;
```

3 - Alterando CozinhaModel
Agora, vamos adicionar a mesma anotação nas propriedades de nosso modelo

```
@ApiModelProperty(example = "1")
@JsonView(RestauranteView.Resumo.class)
private Long id;

@ApiModelProperty(example = "Brasileira")
@JsonView(RestauranteView.Resumo.class)
private String nome;
```

4 - Criando CozinhaControllerOpenApi

```
@Api(tags = "Cozinhas")
public interface CozinhaControllerOpenApi {

  @ApiOperation("Lista as cozinhas com paginação")
  public Page<CozinhaModel> listar(Pageable pageable);

  @ApiOperation("Busca uma cozinha por ID")
  @ApiResponses({
    @ApiResponse(code = 400, message = "ID da cozinha inválido", response =
Problem.class),
    @ApiResponse(code = 404, message = "Cozinha não encontrada", response =
Problem.class)
  })
  public CozinhaModel buscar(
      @ApiParam(value = "ID de uma cozinha", example = "1")
      Long cozinhaId);

  @ApiOperation("Cadastra uma cozinha")
  @ApiResponses({
    @ApiResponse(code = 201, message = "Cozinha cadastrada"),
  })
  public CozinhaModel adicionar(
      @ApiParam(name = "corpo", value = "Representação de uma nova cozinha")
      CozinhaInput cozinhaInput);

  @ApiOperation("Atualiza uma cozinha por ID")
  @ApiResponses({
    @ApiResponse(code = 200, message = "Cozinha atualizada"),
    @ApiResponse(code = 404, message = "Cozinha não encontrada", response =
Problem.class)
  })
  public CozinhaModel atualizar(
      @ApiParam(value = "ID de uma cozinha", example = "1")
      Long cozinhaId,

      @ApiParam(name = "corpo", value = "Representação de uma cozinha com os
novos dados")
      CozinhaInput cozinhaInput);

  @ApiOperation("Exclui uma cozinha por ID")
  @ApiResponses({
    @ApiResponse(code = 204, message = "Cozinha excluída"),
    @ApiResponse(code = 404, message = "Cozinha não encontrada", response =
Problem.class)
  })
  public void remover(
      @ApiParam(value = "ID de uma cozinha", example = "1")
      Long cozinhaId);
}
```

5 - Alterando CozinhaController
Vamos alterar CozinhaController para que o mesmo implemente a interface CozinhaControllerOpenApi

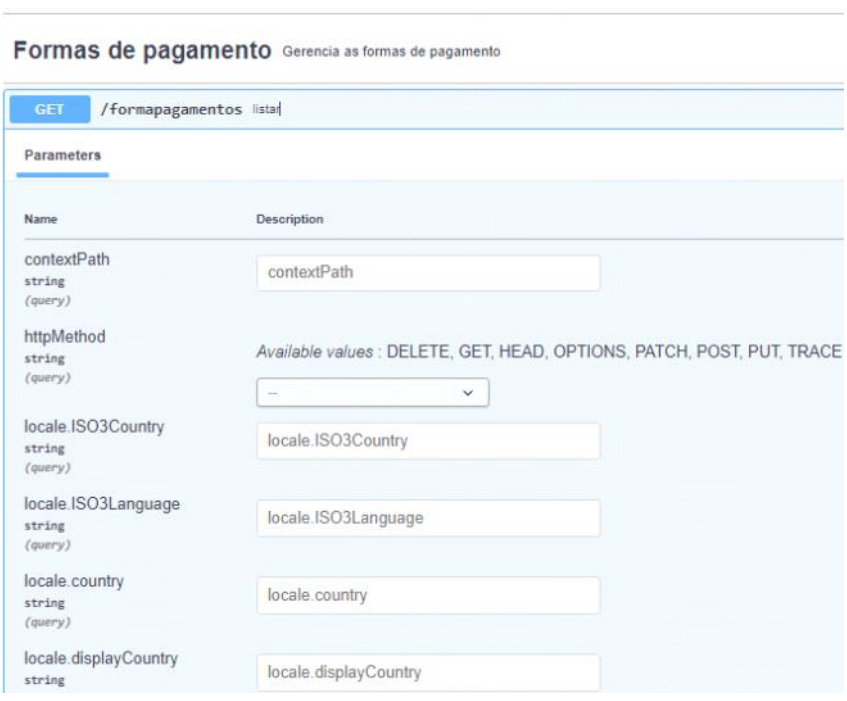
```
@RestController
@RequestMapping(value = "/cozinhas", produces =
MediaType.APPLICATION_JSON_VALUE)
public class CozinhaController implements CozinhaControllerOpenApi {
  //...
}
```

18.23. Ignorando tipos de parâmetros de operações na documentação

segunda-feira, 10 de abril de 2023 22:39

- Na aula 17.9. Implementando requisições condicionais com Deep Etags tivemos que captura uma instância de Servlet para utilizar os elementos da requisição para implementar o cache etag customizado, porém, o SpringFox mapeia os elementos do Servlet como se fossem parâmetros de requisição

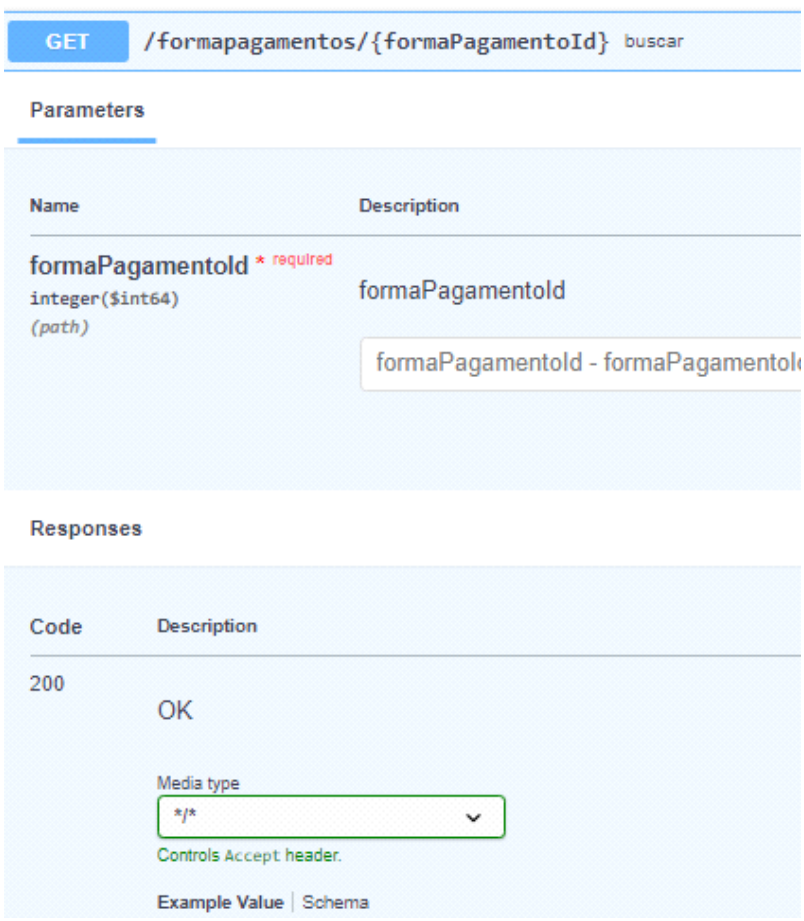
```
@GetMapping("/{formaPagamentoId}")
public ResponseEntity<FormaPagamentoDTO> buscar(@PathVariable Long formaPagamentoId, ServletWebRequest request){
    final FormaPagamento formaPagamento = formaPagamentoService.buscar(formaPagamentoId);
    ShallowEtagHeaderFilter.disableContentCaching(request.getRequest());
```



ou seja, tempos que ignorar a instância de ServletWebRequest no parâmetro do método controlador

```
private Class[] ignoredParameterTypes() {

    return new Class[]{
            Cidade.class,
            Cozinha.class,
            Endereco.class,
            FormaPagamento.class,
            Produto.class,
            Restaurante.class,
            Usuario.class,
            VendaDiaria.class,
            ServletWebRequest.class};
}
```

```
public Docket apiDocket() {

    return new Docket(DocumentationType.OAS_30)
            .ignoredParameterTypes(ignoredParameterTypes())
            .select()
            //.apis(RequestHandlerSelectors.any())/*Todos os end
```



18.24. Desafio: descrevendo documentação de endpoints de formas de pagamento

terça-feira, 11 de abril de 2023 08:28

1 - Alterando SpringFoxConfig
Primeiro, vamos alterar o método apiDocket, adicionando novas Tags

```
.tags(new Tag("Cidades", "Gerencia as cidades"),
      new Tag("Grupos", "Gerencia os grupos de usuários"),
      new Tag("Cozinhas", "Gerencia as cozinhas"),
      new Tag("Formas de pagamento", "Gerencia as formas de pagamento"));
```

2 - Alterando FormaPagamentoInput
Vamos adicionar a anotação @ApiModelProperty acima da propriedade "descricao"

```
@ApiModelProperty(example = "Cartão de crédito", required = true)
@NotBlank
private String descricao;
```

3 - Alterando FormaPagamentoModel
Agora, vamos adicionar a mesma anotação nas propriedades de nosso modelo

```
@ApiModelProperty(example = "1")
private Long id;

@ApiModelProperty(example = "Cartão de crédito")
private String descricao;
```

4 - Criando FormaPagamentoControllerOpenApi

```
@Api(tags = "Formas de pagamento")
public interface FormaPagamentoControllerOpenApi {

  @ApiOperation("Lista as formas de pagamento")
  public ResponseEntity<List<FormaPagamentoModel>> listar(ServletWebRequest
request);

  @ApiOperation("Busca uma forma de pagamento por ID")
  @ApiResponses({
    @ApiResponse(code = 400, message = "ID da forma de pagamento inválido",
response = Problem.class),
    @ApiResponse(code = 404, message = "Forma de pagamento não encontrada",
response = Problem.class)
  })
  public ResponseEntity<FormaPagamentoModel> buscar(
      @ApiParam(value = "ID de uma forma de pagamento", example = "1")
      Long formaPagamentoId,

      ServletWebRequest request);

  @ApiOperation("Cadastra uma forma de pagamento")
  @ApiResponses({
    @ApiResponse(code = 201, message = "Forma de pagamento cadastrada"),
  })
  public FormaPagamentoModel adicionar(
      @ApiParam(name = "corpo", value = "Representação de uma nova forma de
pagamento")
      FormaPagamentoInput formaPagamentoInput);

  @ApiOperation("Atualiza uma cidade por ID")
  @ApiResponses({
    @ApiResponse(code = 200, message = "Forma de pagamento atualizada"),
    @ApiResponse(code = 404, message = "Forma de pagamento não encontrada",
response = Problem.class)
  })
  public FormaPagamentoModel atualizar(
      @ApiParam(value = "ID de uma forma de pagamento", example = "1")
      Long formaPagamentoId,

      @ApiParam(name = "corpo", value = "Representação de uma forma de pagamento
com os novos dados")
      FormaPagamentoInput formaPagamentoInput);

  @ApiOperation("Exclui uma forma de pagamento por ID")
  @ApiResponses({
    @ApiResponse(code = 204, message = "Forma de pagamento excluída"),
    @ApiResponse(code = 404, message = "Forma de pagamento não encontrada",
response = Problem.class)
  })
  public void remover(Long formaPagamentoId);
}
```

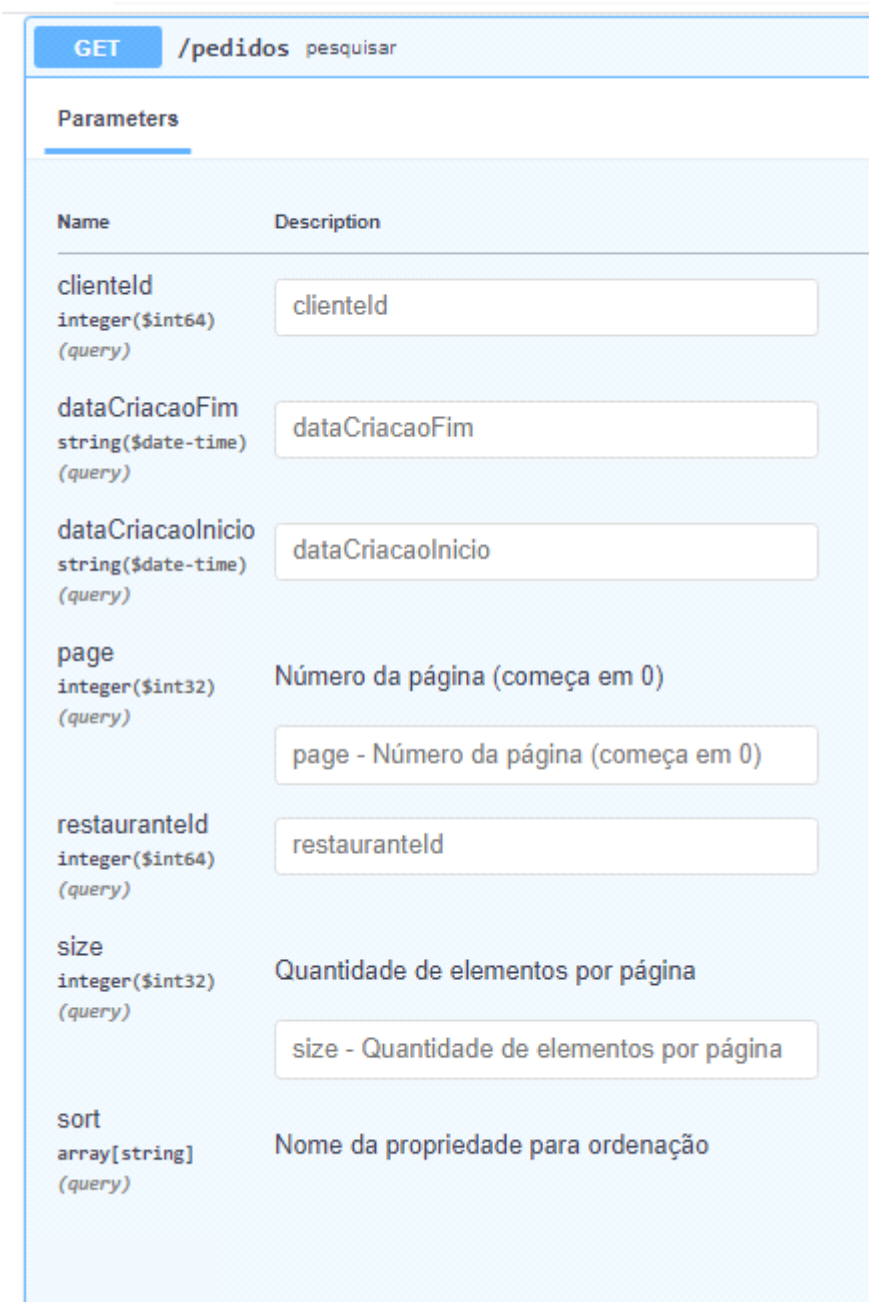
5 - Alterando FormaPagamentoController
Vamos alterar FormaPagamentoController para que o mesmo implemente a interface FormaPagamentoControllerOpenApi

```
@RestController
@RequestMapping(path = "/formas-pagamento", produces =
MediaType.APPLICATION_JSON_VALUE)
public class FormaPagamentoController implements
FormaPagamentoControllerOpenApi {
  //...
}
```

18.25. Descrevendo parâmetros globais em operações

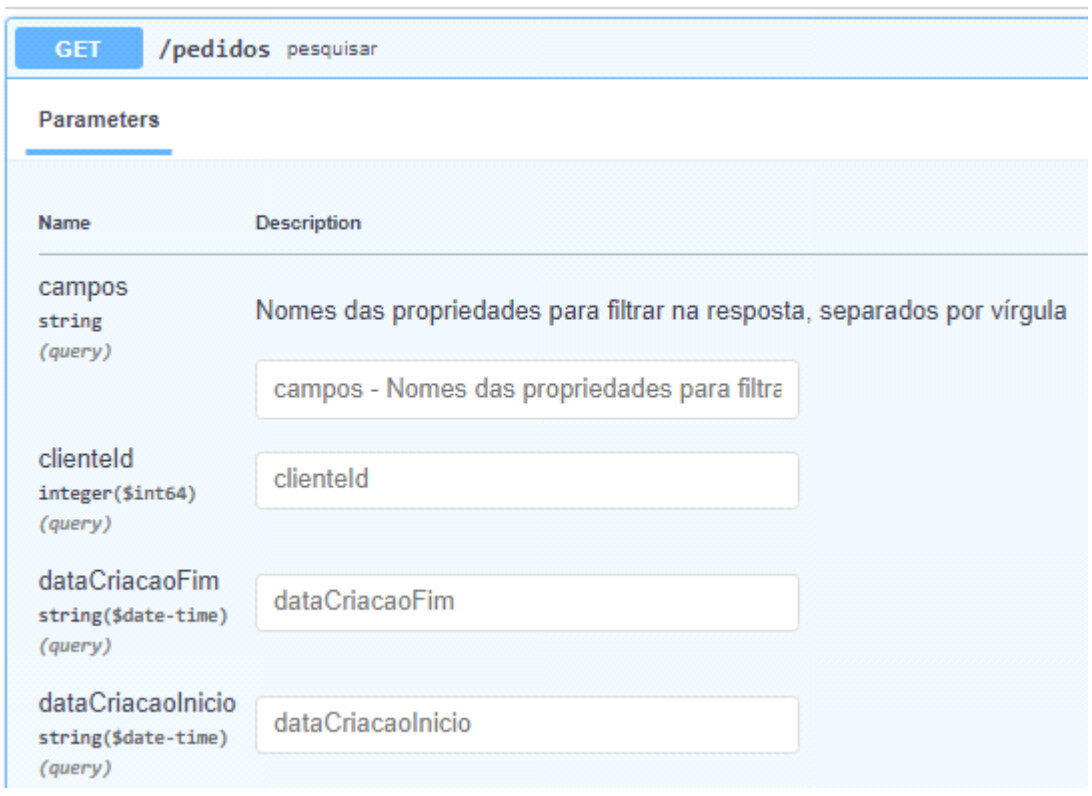
terça-feira, 11 de abril de 2023 08:57

- Na aula 13.3. Limitando os campos retornados pela API com Squiggly adicionando a dependência para filtrar quais campos queríamos na resposta da requisição do endpoint de pedidos de pedidos, porém, na documentação isso não é mapeado porque não explicitamos o filtro do Squiggly no controlador .



- Adicionar o parâmetro de forma global na documentação

```
)
.globalRequestParameters(Collections.singletonList(
        new RequestParameterBuilder()
                .name("campos")
                .description("Nomes das propriedades para filtrar na resposta, separados por vírgula")
                .in(ParameterType.QUERY)
                .required(false)
                .query(q -> q.model(m -> m.scalarModel(ScalarType.STRING)))
                .build()));
```



18.26. Descrevendo parâmetros implícitos em operações

terça-feira, 11 de abril de 2023 10:22

No Squiggly, configuramos apenas para endpoints específicos

```java
@Configuration
public class SquigglyConfig {

    @Bean
    public FilterRegistrationBean<SquigglyRequestFilter> squigglyRequestFilter(ObjectMapper objectMapper){
        Squiggly.init(objectMapper, new RequestSquigglyContextProvider( filterParam: "campos", defaultFilter: null));

        /*especificando endpoints ativos para a QueryParams - fields*/
        var urlParams :List<String> = Arrays.asList("/pedidos/*","/restaurantes/*");

        final FilterRegistrationBean<SquigglyRequestFilter> filterRegistrationBean = new FilterRegistrationBean<~>();
        filterRegistrationBean.setFilter(new SquigglyRequestFilter());
        filterRegistrationBean.setOrder(1);
        filterRegistrationBean.setUrlPatterns(urlParams);
        return filterRegistrationBean;
    }
}
```

- Endpoint pedido: método buscar e listar

```java
@ApiImplicitParams({
        @ApiImplicitParam(value = "Nomes das propriedades para filtrar na resposta, separados por vírgula"
        , name = "campos", paramType = "query", type = "string")
})
@GetMapping
public Page<PedidoResumoDTO> pesquisar(PedidoFilter pedidoFilter, Pageable pageable){
    pageable = truduzirPageable(pageable);
    final Page<Pedido> pedidosPage = pedidoService.pesquisar(pedidoFilter, pageable);
    final List<PedidoResumoDTO> pedidosResumoDTO = pRAssembler.toListDTO(pedidosPage.getContent());
    final Page<PedidoResumoDTO> pedidoResumoDTOS = new PageImpl<>(pedidosResumoDTO, pageable, pedidosPage.getTotalElements());
    return pedidoResumoDTOS;
}
```

```java
@ApiImplicitParams({
        @ApiImplicitParam(value = "Nomes das propriedades para filtrar na resposta, separados por vírgula"
                , name = "campos", paramType = "query", type = "string")
})
@GetMapping("/{codigoId}")
public PedidoDTO buscar(@PathVariable String codigoId) { return pAssembler.toDTO(pedidoService.buscar(codigoId)); }
```

Agora, na documentação, somente endpoints de e Pedido e nos métodos listar e buscar foram adicionados as query params

18.27. Desafio: descrevendo documentação de endpoints de pedidos

terça-feira, 11 de abril de 2023 10:42

18.28. Descrevendo parâmetros de projeções em endpoints de consultas

terça-feira, 11 de abril de 2023 10:46

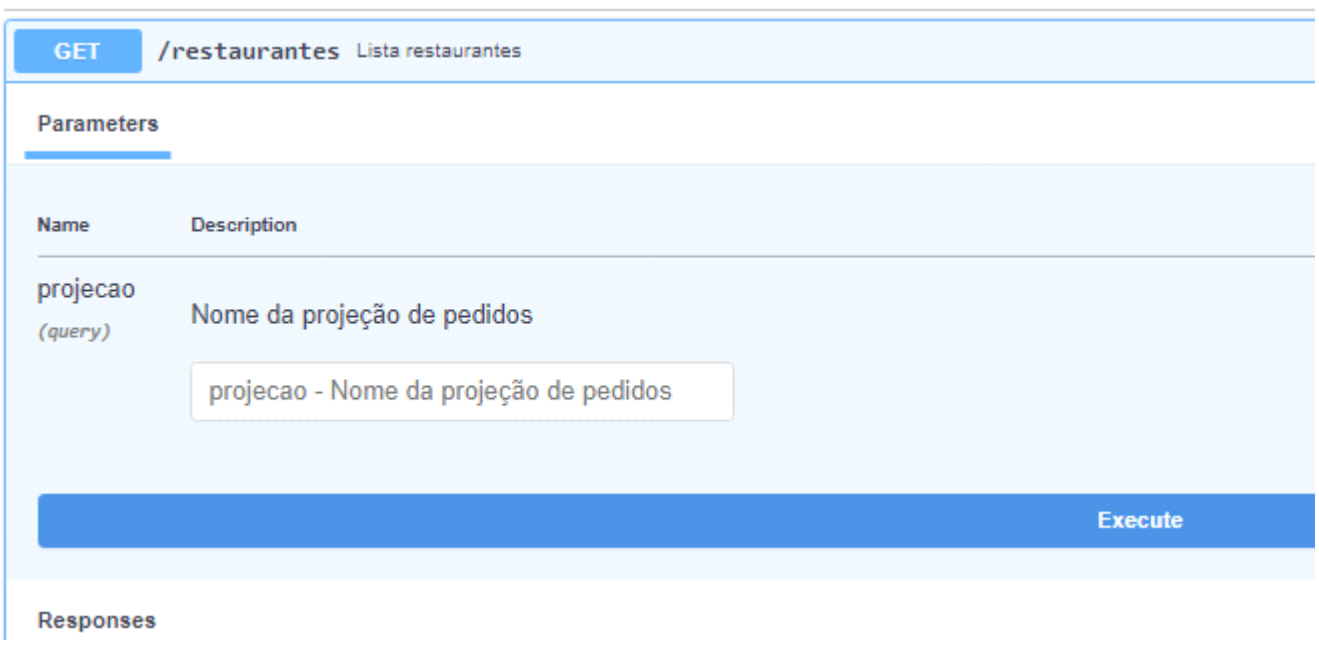
- Na documentação usando SpringFox, uma coisa substitui outra quando há nomes iguais

Na listagem de restaurantes há apenas um endpoint de listagem, porém, há 2 endpoints, mas o que diferencia são os parâmetros da requisição

```java
@JsonView(RestauranteView.Resumo.class)
@GetMapping
public List<RestauranteDTO> listar() {
    final List<Restaurante> restaurantes = restauranteService.listar();

    return rAssembler.toListDTO(restaurantes);
}

@JsonView(RestauranteView.ResumoApenasNome.class)
@GetMapping(params = "projecao=apenas-nome")
public List<RestauranteDTO> listarApenasNome() {

    return listar();
}
```

podemos alterar para listar sem projeção por padrão

```java
@ApiOperation(value = "Lista restaurantes")
@ApiImplicitParams({
        @ApiImplicitParam(value = "Nome da projeção de pedidos", allowableValues = "apenas-nome",
                name = "projecao", paramType = "query", type = "string")
})
@JsonView(RestauranteView.Resumo.class)
@GetMapping
public List<RestauranteDTO> listar() {
    final List<Restaurante> restaurantes = restauranteService.listar();

    return rAssembler.toListDTO(restaurantes);
}

@ApiOperation(value = "Lista restaurantes", hidden = true)
@JsonView(RestauranteView.ResumoApenasNome.class)
@GetMapping(params = "projecao=apenas-nome")
public List<RestauranteDTO> listarApenasNome() {

    return listar();
}
```



18.29. Desafio: descrevendo documentação de endpoints de restaurantes

terça-feira, 11 de abril de 2023 15:02