

13.1. Fazendo projeção de recursos com @JsonView do Jackson

segunda-feira, 27 de março de 2023 15:03

Até o momento estamos usando DTO's para a representação do recurso, mas podemos customizar mais ainda a representação do DTO usando a anotação JsonView do Jackson.

Podemos criar uma interface para mapear o conjunto de atributos a serem representados.

```
package com.algaworks.algafood.api.model.view;

public interface RestauranteView {
    5 usages
    public interface Resmo{
    }
}
```

Na representação de um Restaurante, não queremos visualizar informações que não necessariamente precisam estar em uma coleção de recursos. Podemos anotar os atributos que queremos representar na resposta:

```
12 public class RestauranteDTO {
13
14     @JsonView(RestauranteView.Resmo.class)
15     private Long id;
16     @JsonView(RestauranteView.Resmo.class)
17     private String nome;
18     @JsonView(RestauranteView.Resmo.class)
19     private BigDecimal taxaFrete;
20     @JsonView(RestauranteView.Resmo.class)
21     private CozinhaDTO cozinha;
22     private Boolean ativo = Boolean.TRUE;
23     private Boolean aberto;
24     private EnderecoDTO endereco;
25 }
26
```

e anotando o método controlador

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

    return rAssembler.toListDTO(restaurantes);
}
```

```

{
  "id": 1,
  "nome": "Thai Gourmet",
  "taxaFrete": 10.00,
  "cozinha": {}
},
{
  "id": 2,
  "nome": "Thai Delivery",
  "taxaFrete": 9.50,
  "cozinha": {}
},
{
  "id": 3,
  "nome": "Tuk Tuk Comida Indiana",
  "taxaFrete": 15.00,
  "cozinha": {}
},
{
  "id": 4,

```

Caso haja objetos anotados com JsonView, para representar os atributos dentro dos objetos, os mesmo deverão ser anotados com JsonView

```

{
  "id": 1,
  "nome": "Thai Gourmet",
  "taxaFrete": 10.00,
  "cozinha": {
    "id": 1,
    "nome": "Tailandesa"
  }
},

```

Podemos criar outro controlador para representar a chamada de uma requisição "resumida" utilizamos parâmetros de requisição.

GET ⌵ http://localhost:8080/restaurantes?projecao=resumo

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	projecao	resumo
	Key	Value

Podemos criar mais uma interface para representar a visualização de recursos com apenas Nome e Id na listagem.

```

1 package com.algaworks.algafood.api.model.view;
2
3 public interface RestauranteView {
4     8 usages
5     public interface Resumo {}
6     2 usages
7     public interface ResumoApenasNome {}
8 }

```

Criar mais um método controlador

```

@JsonView({RestauranteView.Resumo.class, RestauranteView.ResumoApenasNome.class})
private Long id;
@JsonView({RestauranteView.Resumo.class, RestauranteView.ResumoApenasNome.class})
private String nome;
@JsonView(RestauranteView.Resumo.class)
private BigDecimal taxaFrete;
@JsonView(RestauranteView.Resumo.class)

```

A anotação JsonView também aceita um array de classes

```

12 public class RestauranteDTO {
13
14     @JsonView({RestauranteView.Resumo.class, RestauranteView.ResumoApenasNome.class})
15     private Long id;
16     @JsonView({RestauranteView.Resumo.class, RestauranteView.ResumoApenasNome.class})
17     private String nome;
18     @JsonView(RestauranteView.Resumo.class)
19     private BigDecimal taxaFrete;
20     @JsonView(RestauranteView.Resumo.class)
21     private CozinhaDTO cozinha;
22     private Boolean ativo = Boolean.TRUE;
23     private Boolean aberto;
24     private EnderecoDTO endereco;
25 }

```

13.2. Limitando os campos retornados pela API com @JsonFilter do Jackson

terça-feira, 28 de março de 2023 15:28

Limitar a representação do recurso baseado em parâmetros especificados pelo consumidor da API. É diferente do que foi usado na aula 13.1, pois foi utilizado Query Params na requisição. O consumidor pode especificar quais propriedades ele quer na representação de determinado recurso.

Retornar campos especificados pelo consumidor da API em uma requisição utilizando JsonFilter.

Primeiro, é necessário anotar a classe que irá conter o filtro de serialização:

```
@JsonFilter("pedidosFilter")
@Getter
@Setter
public class PedidoResumoDTO {
    private String codigo;
    private BigDecimal subTotal;
    private BigDecimal taxaFrete;
    private BigDecimal valorTotal;
    private OffsetDateTime dataCriacao;
    private String status;
    private RestauranteResumoDTO restaurante;
    private UsuarioDTO cliente;
}
```

e fazer a configuração do filtro no controlador.

a ideia é adicionar um campo no parâmetro da requisição que conterá as propriedades que o consumidor da API desejar:

GET http://localhost:8080/pedidos?campos=codigo, valorTotal, restaurante

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	campos	codigo, valorTotal, restaurante
	Key	Value

e ser capturada no controlador:

```
@GetMapping
public MappingJacksonValue listar(@RequestParam String campos){
    final List<PedidoResumoDTO> pedidos = pRAssembler.toListDTO(pedidoService.listar());

    MappingJacksonValue mappingJacksonValue = new MappingJacksonValue(pedidos);
    SimpleFilterProvider simpleFilterProvider = new SimpleFilterProvider();
    simpleFilterProvider.addFilter( id: "pedidosFilter", SimpleBeanPropertyFilter.serializeAll());
    if(StringUtils.isNotBlank(campos)){
        simpleFilterProvider.addFilter( id: "pedidosFilter", SimpleBeanPropertyFilter.filterOutAllExcept(campos.split( regex: ",")));
    }

    mappingJacksonValue.setFilters(simpleFilterProvider);
    return mappingJacksonValue;
}
```

Para retornar uma lista de pedidos, primeiro trocamos o tipo de retorno do método para MappingJacksonValue.

- linha 47: criação de um objeto MappingJacksonValue de pedidos
- linha 48: criação de um filtro com SimpleFilterProvider
- linha 49: adicionar filtros anotados com @JsonFilter e as propriedades a serem filtradas/não serializadas, no caso, é possível usar a classe utilitária SimpleBeanPropertyFilter chamando o método serializeAll para serializar todas as propriedades da classe anotada filtrada de nome "pedidosFilter". A lógica de negócio é por padrão (sem parâmetros na requisição) serializar todas as propriedades do recurso.
- linha 50: caso a string campos não seja nula, adicionar um filtro que pode ser substituído. o 1º argumento recebe o nome da classe filtrada e o 2º argumento recebe um SimpleBeanPropertyFilter chamando filterOutAllExcep(filtrarTodasExceto) que recebe um array de String contendo o nome das propriedades dentro de campos.
- linha 55: é necessário adicionar o filtro dentro do objeto mappingJacksonValue que contém a lista de pedidos.

13.3. Limitando os campos retornados pela API com Squiggly

quarta-feira, 29 de março de 2023 23:07

```
Pom.xml
<dependency>
<groupid>com.github.bohnman</groupid>
<artifactId>squiggly-filter-jackson</artifactId>
<version>1.3.18</version>
</dependency>
```

O Squiggly Filter é um Jackson JSON PropertyFilter, que seleciona as propriedades de um objeto/lista/mapa usando um subconjunto da sintaxe de filtragem da API do Facebook Graph .

A ideia é deixar o consumidor da API especificar as propriedades desejadas do recurso na representação do recurso, como estávamos fazendo anteriormente. Porém, diferente das nossas implementações, o Squiggly mapeia todas as requisições com um Servlet, e não somente a classe implementada.

```
13  @Configuration
14  public class SquigglyConfig {
15
16      @Bean
17      public FilterRegistrationBean<SquigglyRequestFilter> squigglyRequestFilter(ObjectMapper objectMapper){
18          Squiggly.init(objectMapper, new RequestSquigglyContextProvider( filterParam: "campos", defaultFilter: null));
19
20          /*especificando endpoints ativos para a QueryParams - fields*/
21          var urlParams :List<String> = Arrays.asList("/pedidos/*","/restaurantes/*");
22
23          final FilterRegistrationBean<SquigglyRequestFilter> filterRegistrationBean = new FilterRegistrationBean<>();
24          filterRegistrationBean.setFilter(new SquigglyRequestFilter());
25          filterRegistrationBean.setOrder(1);
26          filterRegistrationBean.setUrlPatterns(urlParams);
27          return filterRegistrationBean;
28      }
```

GET http://localhost:8080/pedidos?fields=codigo,valorTotal,sub*,cliente[id,nome],restaurante[-id]

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input type="checkbox"/>	fields	codigo,valorTotal,sub*,cliente[id,nome]
<input type="checkbox"/>	fields	-cliente
<input checked="" type="checkbox"/>	fields	codigo,valorTotal,sub*,cliente[id,nome],restaurante[-id]
	Key	Value

13.4. Implementando pesquisas simples na API

quinta-feira, 30 de março de 2023

18:21

Até o momento, na representação de produtos de um restaurante, estamos trazendo todos os produtos, sejam ativos ou inativos. Na aula vamos implementar a busca por produtos que sejam somente ativos.

- Implementar uma query JPQL para a busca de produtos somente ativos associados a um restaurante

```
Usage
@Query("from Produto p where p.ativo = true and p.restaurante = :restaurante")
List<Produto> findByAtivosRestaurante(Restaurante restaurante);
```

- Alterar o endpoint que lista os produtos para receber uma **Query Param** da requisição, um valor booleano que especifica incluir produtos inativos.

```
@GetMapping
public List<ProdutoDTO> listar(@PathVariable Long restauranteId,
                               @RequestParam (required = false) boolean incluirInativos){
    final Restaurante restaurante = restauranteService.buscar(restauranteId);
    List<Produto> produtos = null;

    if(incluirInativos){
        produtos = produtoService.listarTodosPorRestaurante(restaurante);
    } else {
        produtos = produtoService.listarAtivosPorRestaurante(restaurante);
    }
}
```

13.5. Modelando pesquisas complexas na API

quinta-feira, 30 de março de 2023

21:37

Tornar a pesquisa de recursos dinâmicas com filtros para a resposta pode ser complexo, em relação à pesquisa simples vista na aula 13.4. Nesta aula é apresentado algumas soluções para pesquisas complexas, por exemplo, a data de criação de um pedido ou todos os pedidos a partir de uma data, ou pelo cliente ou emitido pelo restaurante.

- Receber parâmetros de URL no recurso da coleção GET OK

REQUISIÇÃO

```
GET /pedidos?dataCriacaoInicio=2019-10-20T14:00:00Z
&dataCriacaoFim=2019-10-30T14:00:00Z&restauranteId=1&clienteId=2
```

RESPOSTA

```
HTTP/1.1 200 OK
```

```
[
  {
    "codigo": "f9981ca4",
    "valorTotal": 308.90,
    "status": "CRIADO"
  },
  {
    "codigo": "d1083ba8",
    "valorTotal": 103.40,
    "status": "CONFIRMADO"
  }
]
```

- Considerar a própria pesquisa como um recurso POST OK

REQUISIÇÃO

```
POST /pedidos/filtros
{
  "dataCriacaoInicio": "2019-10-20T14:00:00Z",
  "dataCriacaoFim": "2019-10-30T14:00:00Z",
  "restauranteId": 1,
  "clienteId": 2
}
```


RESPOSTA

HTTP/1.1 200 OK

```
[
  {
    "codigo": "f9981ca4",
    "valorTotal": 308.90,
    "status": "CRIADO"
  },
  {
    "codigo": "d1083ba8",
    "valorTotal": 103.40,
    "status": "CONFIRMADO"
  }
]
```

Não é considerado uma boa prática fazer uma requisição POST não adicionando um recurso, fere os parâmetros de uma aplicação RESTful

- Considerar a própria pesquisa como um recurso (de verdade) POST CREATED
 - Criar um objeto que simule um filtro para ser usado em uma busca pelo recurso. a requisição POST cria um filtro com as propriedades passadas no corpo.

REQUISIÇÃO

```
POST /pedidos/filtros
{
  "dataCriacaoInicio": "2019-10-20T14:00:00Z",
  "dataCriacaoFim": "2019-10-30T14:00:00Z",
  "restauranteId": 1,
  "clienteId": 2
}
```

RESPOSTA

HTTP/1.1 201 Created

```
{
  "id": 234,
  "dataCriacaoInicio": "2019-10-20T14:00:00Z",
  "dataCriacaoFim": "2019-10-30T14:00:00Z",
  "restauranteId": 1,
  "clienteId": 2
}
```

O id do filtro é utilizado na busca de um recurso com as propriedades do filtro

REQUISIÇÃO

```
GET /pedidos/filtros/234
```


RESPOSTA

```
HTTP/1.1 200 OK

{
  "id": 234,
  "dataCriacaoInicio": "2019-10-20T14:00:00Z",
  "dataCriacaoFim": "2019-10-30T14:00:00Z",
  "restauranteId": 1,
  "clienteId": 2,
  "resultado": [
    {
      "codigo": "f9981ca4",
      "valorTotal": 308.90,
      "status": "CRIADO"
    },
    {
      "codigo": "d1083ba8",
      "valorTotal": 103.40,
      "status": "CONFIRMADO"
    }
  ]
}
```

- Usa o Id do filtro criado como parâmetro de URL da coleção do recurso POST CREATED

REQUISIÇÃO

```
GET /pedidos?filtro=234
```

RESPOSTA

```
HTTP/1.1 200 OK

[
  {
    "codigo": "f9981ca4",
    "valorTotal": 308.90,
    "status": "CRIADO"
  },
  {
    "codigo": "d1083ba8",
    "valorTotal": 103.40,
    "status": "CONFIRMADO"
  }
]
```

- Quando os critérios de pesquisa são realmente complexos
Para casos no qual a apresentam operadores lógicos como mecanismo de filtragem de coleção de recursos.

REQUISIÇÃO

```
POST /pedidos/filtros
[
  {
    "operadorLogico": "_ou",
    "critérios": [
      "dataInicio": {
        "operadorIgualdade": "maiorQue",
        "valor": "2019-10-20T14:00:00Z"
      },
      "dataFim": {
        "operadorIgualdade": "menorQue",
        "valor": "2019-10-30T14:00:00Z"
      }
    ]
  },
  {
    "operadorLogico": "e",
    "critérios": [
      "restauranteId": {
        "operadorIgualdade": "igual",
        "valor": 1
      },
      "clienteId": {
        "operadorIgualdade": "igual",
        "valor": 10
      }
    ]
  }
]
```

RESPOSTA

HTTP/1.1 201 Created

```
{
  "id": 234,
  ...
}
```

REQUISIÇÃO

```
GET /pedidos?filtro=234
```

RESPOSTA

HTTP/1.1 200 OK

```
[
  {
    "codigo": "f9981ca4",
    "valorTotal": 308.90,
    "status": "CRIADO"
  },
  {
    "codigo": "d1083ba8",
    "valorTotal": 103.40,
    "status": "CONFIRMADO"
  }
]
```

13.6. Implementando pesquisas complexas na API

quinta-feira, 30 de março de 2023 22:36

- Adicionar mais dados na massa de testes
 - Adicionado pedidos e item de pedidos
- Criar classes para representar o filtro de pesquisa complexa
 - Na verdade é um DTO e não existe convenção de nome para tal filtro
 - pacote -> domain.repository.filter

Primeiro criamos uma classe para fabricar as especificações (SpecificationFactory). Como visto na aula [5.18. Criando uma fábrica de Specifications](#)

```
2 usages
public class PedidoSpecs {

    1 usage
    public static Specification<Pedido> filtroPedidoSpec (PedidoFilter pedidoFilter){
        return ((root, query, criteriaBuilder) -> {
            List<Predicate> predicates = new ArrayList<>();

            if(pedidoFilter.getClientId() != null){
                predicates.add(criteriaBuilder.equal(root.get("cliente"), pedidoFilter.getClientId()));
            }

            final Predicate[] arrayPredicate = predicates.toArray(new Predicate[0]);

            return criteriaBuilder.and(arrayPredicate);
        });
    }
}
```

No repositório da entidade que está sendo utilizada, neste caso, Pedido, estender a interface JpaSpecificationExecutor<?> para usar os predicates criados.

```
2 usages
@Repository
public interface PedidoRepository extends CustomJpaRepository<Pedido, Long>,
    JpaSpecificationExecutor<Pedido> {

    1 usage
    Optional<Pedido> findByCodigo(String codigo);

    @Query("from Pedido as p join fetch p.cliente join fetch p.restaurante r join fetch r.cozinha")
    List<Pedido> findAll();
}
```

da classe de serviço, pode chamar o método do repositório da interface JpaSpecificationExecutor.

```

1 usage
private Pedido <no parameters>
    return p
}
Example<S> example
Example<S> example, Sort sort
Pageable pageable
Example<S> example, Pageable pageable
@Nullable Specification<Pedido> spec
@Nullable Specification<Pedido> spec, Pageable pageable
1 usage
@Nullable Specification<Pedido> spec, Sort sort
public List<Pedido> pesquisar(PedidoFilter pedidoFilter) {
    return pedidoRepository.findAll(PedidoSpecs.filtroPedidoSpec(pedidoFilter));
}

```

O objeto PedidoFilter pode ser criado com parâmetros na requisição e ser capturado no método controlador

```

@GetMapping
public List<PedidoResumoDTO> pesquisar(PedidoFilter pedidoFilter){
    final List<PedidoResumoDTO> pedidos = pAssembler.toListDTO(pedidoService.pesquisar(pedidoFilter));
}

```

Dentro da classe PedidoFilter, temos atributos para filtragem de Pedidos.

```

@Setter
@Getter
public class PedidoFilter {

    private Long clienteId;
    private Long restauranteId;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    private OffsetDateTime dataCriacaoInicio;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    private OffsetDateTime dataCriacaoFim;
}

```

@DateTimeFormat: Declara que um parâmetro de campo ou método deve ser formatado como uma data ou hora

é necessário anotar as classes para que a conversão de um campo de data/hora do parâmetro da requisição seja realizado para OffsetDateTime

A requisição fica a seguinte:

GET ▼ http://localhost:8080/pedidos?clienteld=1&dataCriacaoInicio=2019-09-02T00:00:00Z&dataCriacaoFim=2019-10-30T23:59:59Z

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	clienteld	1
<input type="checkbox"/>	restauranteld	4
<input checked="" type="checkbox"/>	dataCriacaoInicio	2019-09-02T00:00:00Z
<input checked="" type="checkbox"/>	dataCriacaoFim	2019-10-30T23:59:59Z

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```

1  {
2    {
3      "codigo": "3c87dd3e-93b5-4fad-a5fb-5bb3ad4e092d",
4      "subTotal": 79.00,
5      "taxaFrete": 0.00,
6      "valorTotal": 79.00,
7      "dataCriacao": "2019-09-02T21:00:30Z",
8      "status": "Criado",
9      "restaurante": {
10       "id": 4,
11       "nome": "Java Steakhouse"
12     },
13     "cliente": {
14       "id": 1,
15       "nome": "João da Silva",
16       "email": "joao.ger@algafood.com"
17     }
18   },
19   {
20     "codigo": "b5741512-8fbc-47fa-9ac1-b530354fc0ff",
21     "subTotal": 110.00,
22     "taxaFrete": 10.00,
23     "valorTotal": 120.00,
24     "dataCriacao": "2019-10-30T21:10:00Z",
25     "status": "Entregue",
26     "restaurante": {
27       "id": 1,
28       "nome": "Thai Gourmet"
29     },
30     "cliente": {

```

Para adicionar outros filtros, basta apenas adicionar condições para que o atributo não seja nulo, atribuir predicados:

```

public static Specification<Pedido> filtroPedidoSpec (PedidoFilter pedidoFilter){
    return ((root, query, criteriaBuilder) -> {
        List<Predicate> predicates = new ArrayList<>();

        if(pedidoFilter.getClientId() != null){
            predicates.add(criteriaBuilder.equal(root.get("cliente"), pedidoFilter.getClientId()));
        }

        if(pedidoFilter.getRestauranteId() != null){
            predicates.add(criteriaBuilder.equal(root.get("restaurante"), pedidoFilter.getRestauranteId()));
        }

        if(pedidoFilter.getDataCriacaoInicio() != null){
            predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("dataCriacao"), pedidoFilter.getDataCriacaoInicio()));
        }

        if(pedidoFilter.getDataCriacaoFim() != null){
            predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get("dataCriacao"), pedidoFilter.getDataCriacaoFim()));
        }

        final Predicate[] arrayPredicate = predicates.toArray(new Predicate[0]);

        return criteriaBuilder.and(arrayPredicate);
    });
}

```

Nota: acaba gerando o problema do N + 1. Para diminuir as consultas ao banco, o root possui métodos de fetch.

```

return ((root, query, criteriaBuilder) -> {
    root.fetch( attributeName: "restaurante").fetch( attributeName: "cozinha");
    root.fetch( attributeName: "cliente");
    List<Predicate> predicates = new ArrayList<>();
}

```

```

Hibernate: select pedido0_.id as id1_7_, pedido0_.usuario_cliente_id as usuario16_7_, pedido0_.data_cadastro as data_cad2_13_0_, pedido0_.aberto as aberto2_10_0_ from pedido0_
Hibernate: select usuario0_.id as id1_13_0_, usuario0_.data_cadastro as data_cad2_13_0_, usuario0_.aberto as aberto2_10_0_ from usuario0_
Hibernate: select restaurante0_.id as id1_10_0_, restaurante0_.aberto as aberto2_10_0_, restaurante0_.data_cadastro as data_cad2_10_0_ from restaurante0_

```

13.7. Tratando BindException ao enviar parâmetros de URL inválidos

sexta-feira, 31 de março de 2023 13:59

- Sobrescrever handleBindException da classe interceptadora de exceções ApiExceptionHandler

```
5
6  @ControllerAdvice
7  public class ApiExceptionHandler extends ResponseEntityExceptionHandler {
8
    1 usage
```

- Tratar os campos de erros com mensagens customizadas no messages.properties

13.8. Implementando paginação e ordenação em recursos de coleção da API

sexta-feira, 31 de março de 2023 16:22

- Chamar o método findAll no repositório passando um objeto tipo Pageable como argumento.

```
public Page<Cozinha> listar(Pageable pageable){
    return cozinhaRepository.findAll(pageable);
}
```

O retorno do método é do tipo Page tipado com o tipo da entidade do repositório, no caso Page<Cozinha>

- Na camada controller, chamar o método getContent para extrair a lista de entidades da paginação.

```
@GetMapping
public List<CozinhaDTO> listar(Pageable pageable) {
    return cAssembler.toListDTO(cozinhaService.listar(pageable).getContent());
}
```

O próprio Spring faz a instanciação dos argumentos necessários vindos do parâmetro da requisição, no caso de 'Pageable pageable'

GET http://localhost:8080/cozinhas?size=2&page=1

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

	Key	Value	Description
<input checked="" type="checkbox"/>	size	2	Quantidade de itens por página
<input checked="" type="checkbox"/>	page	1	Número da página baseado no total de itens salvos
<input type="checkbox"/>	sort	nome	Ordena pelo atributo 'nome'
<input type="checkbox"/>	sort	id	Ordena pelo atributo 'id'
	Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 2,
4     "nome": "Indiana"
5   },
6   {
7     "id": 1,
8     "nome": "Tailandesa"
9   }
10 }
```

é interessante expor as informações da paginação para o consumidor da API, para que assim, facilite a consulta na API. Logo, basta retornar um objeto tipo Page no retorno da requisição.

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

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

Também é aceitável dentro do repositório, adicionar no 2º argumento um Page para paginar a consulta.

Por padrão, a quantidade de itens por página é 10, mas podemos alterar o tamanho com a anotação @PageableDefault(size=*) no método do controlador

```
@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());
}
```

13.9. Desafio: implementando paginação e ordenação de pedidos

sexta-feira, 31 de março de 2023 18:43

- Adicionar um segundo argumento do método do repositório findAll
 - Como recebe um Specification, a interface JpaSpecificationExecutor possui sobrecargas para objetos Pageable

```
Usage
public Page<Pedido> pesquisar(PedidoFilter pedidoFilter, Pageable pageable) {
    return pedidoRepository.findAll(PedidoSpecs.filtroPedidoSpec(pedidoFilter), pageable);
}
```

- Implementar lógica de conversão de PedidoResumoDTO para um Page de PedidoResumoDTO

```
@GetMapping
public Page<PedidoResumoDTO> pesquisar(PedidoFilter pedidoFilter, Pageable pageable){
    final Page<Pedido> pedidosPage = pedidoService.pesquisar(pedidoFilter, pageable);
    final List<PedidoResumoDTO> pedidosResumoDTO = pRAssembler.toListDTO(pedidosPage.getContent());
    final Page<PedidoResumoDTO> pedidosResumoDTOS = new PageImpl<>(pedidosResumoDTO, pageable, pedidosPage.getTotalElements());
    return pedidosResumoDTOS;
}
```

Nota: Há um bug para fazer a contagem de registros no banco para que o JPA faça a ordenação baseada na paginação e contagem de registros no banco e com o fetch da consulta. Para corrigir, é necessário fazer uma condicional para verificar se o que está sendo buscado de acordo com o fetch é um pedido ou um tipo numérico vindo do COUNT

```
public static Specification<Pedido> filtroPedidoSpec (PedidoFilter pedidoFilter){
    return ((root, query, criteriaBuilder) -> {
        if(Pedido.class.equals(query.getResultType())){
            root.fetch( attributeName: "restaurante").fetch( attributeName: "cozinha");
            root.fetch( attributeName: "cliente");
        }

        List<Predicate> predicates = new ArrayList<>();
    });
}
```

```
Hibernate: select pedido0_.id as id1_7_0_, restaurant1_.id as id1_10_1_, cozinha2_.id as id1_1_2_, usuario0_.id as id1_11_1_ from pedido pedido0_ where pedido0_.usuario_cliente_id=1
Hibernate: select pedido0_.id as id1_7_0_, restaurant1_.id as id1_10_1_, cozinha2_.id as id1_1_2_, usuario0_.id as id1_11_1_ from pedido pedido0_ where pedido0_.usuario_cliente_id=1
Hibernate: select count(pedido0_.id) as col_0_0_ from pedido pedido0_ where pedido0_.usuario_cliente_id=1
```

A query de count é feita ao especificar o size da paginação.

13.10. Implementando JsonSerializer para customizar representação de paginação

sexta-feira, 31 de março de 2023 19:38

A representação de um recurso paginado visto nas aulas anteriores acrescenta propriedades duplicadas e informações desnecessárias que acaba poluindo a representação. Tendo em vista esse agregado visual de informações da paginação, uma classe anotada com `@JsonComponent` e estendida de `JsonSerializer` é capaz de alterar a representação das propriedades Json.

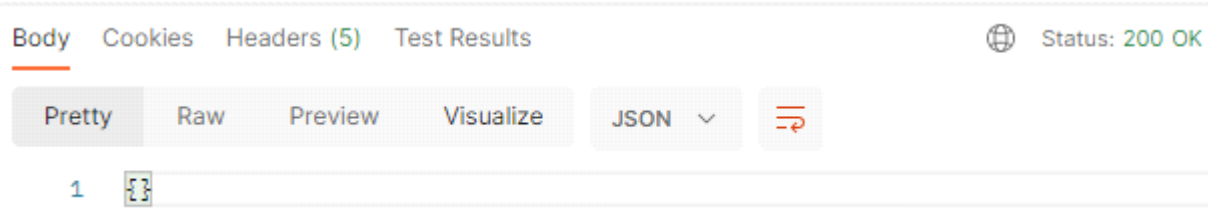
```
package com.algaworks.algafood.core.jackson;

import ...

@JsonComponent
public class PageJsonSerializer extends JsonSerializer<Page<?>> {

    @Override
    public void serialize(Page<?> page, JsonGenerator gen, SerializerProvider serializers) throws IOException {
        gen.writeStartObject();/*Inicio da serialização*/

        gen.writeEndObject();/*Fim da serialização*/
    }
}
```



A resposta é um objeto vazio pois não foi configurado os atributos da serialização.

Com o método `gen.writeObjectField` é possível atribuir um objeto e seu conteúdo dentro da serialização

```
gen.writeStartObject();/*Inicio da serialização*/

gen.writeObjectField( fieldName: "content", page.getContent());

gen.writeEndObject();/*Fim da serialização*/
```

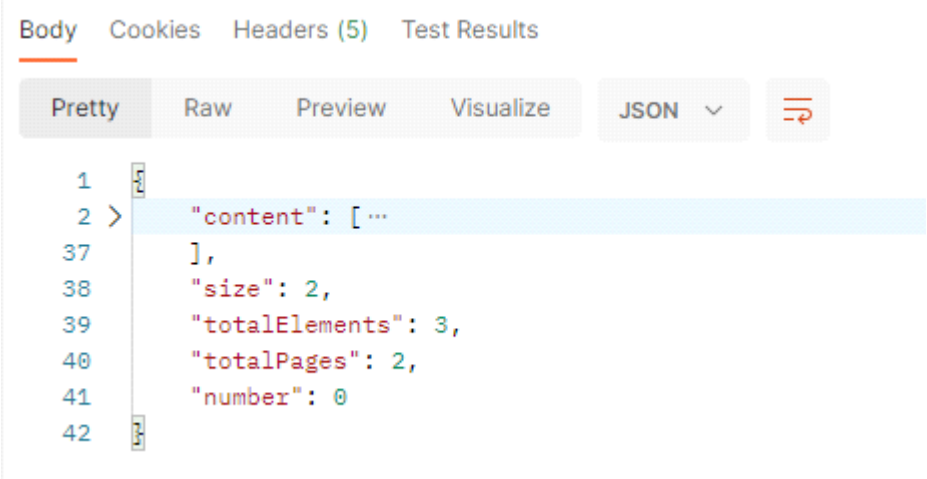


Ainda sem as propriedades de paginação referentes ao page. Para adicionar informações basta escrever o campo e seu objeto associado

```
gen.writeStartObject();/*Inicio da serialização*/

gen.writeObjectField( fieldName: "content", page.getContent());
gen.writeObjectField( fieldName: "size", page.getSize());
gen.writeObjectField( fieldName: "totalElements", page.getTotalElements());
gen.writeObjectField( fieldName: "totalPages", page.getTotalPages());
gen.writeObjectField( fieldName: "number", page.getNumber());

gen.writeEndObject();/*Fim da serialização*/
```



13.11. Implementando um conversor de propriedades de ordenação

sexta-feira, 31 de março de 2023 21:11

Para fazer uma consulta paginada ordenando por nome do cliente, temos que passar no parâmetro da requisição exatamente o nome do atributo na entidade de persistência, e não o nome do atributo da representação do recurso, por exemplo, 'nomeCliente'

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

{

"content": [

{

"codigo": "58bc5f16-2f22-4f94-b2c0-d44ef6a3399d",

"subTotal": 298.90,

"taxaFrete": 10.00,

"valorTotal": 308.90,

"dataCriacao": "2023-04-01T01:33:35Z",

"status": "Criado",

"restaurante": {

"id": 1,

"nome": "Thai Gourmet"

},

"nomeCliente": "João da Silva"

},

],

}

☐

sort

dataCriacao,asc

☒

sort

nomeCliente,desc

Key

Value

gify.Web.SquigglyRequestFilter.doFilter(SquigglyRequestFilter.java:37) <30 internal lines>

936 --- [nio-8080-exec-1] .m.m.a.ExceptionHandlerExceptionResolver : Resolved [org.springframework

g.PropertyReferenceException Create breakpoint : No property 'nomeCliente' found for type 'Pedido'

mapping.PropertyPath.<init>(PropertyPath.java:91)

mapping.PropertyPath.create(PropertyPath.java:438)

Logo, é necessário fazer uma conversão de campos dos parâmetros da requisição

```
private Pageable traduzirPageable (Pageable apiPageable){
    final ImmutableMap<String, String> mapFields = ImmutableMap.of(
        k1: "codigo", v1: "codigo",
        k2: "restaurante.nome", v2: "restaurante.nome",
        k3: "nomeCliente", v3: "cliente.nome",
        k4: "valortotal", v4: "valorTotal"
    );
    return PageableTranslator.translate(apiPageable, mapFields);
}
```

o método 'traduzirPageable' recebe um Pageable somente para delegar para outro método, no corpo do método se cria uma instância de Map com a chave do nome do campo inserido do parâmetro e o valor como o nome do campo na entidade.

```
public static Pageable translate (Pageable pageable, Map<String, String> fieldsMapping){
    var orders :List<Order> = pageable.getSort().stream()
        .filter(order -> fieldsMapping.containsKey(order.getProperty()))
        .map(order -> new Sort.Order(order.getDirection(),
            fieldsMapping.get(order.getProperty())))
        .collect(Collectors.toList());
    return PageRequest.of(pageable.getPageNumber(), pageable.getPageSize(), Sort.by(orders));
}
```

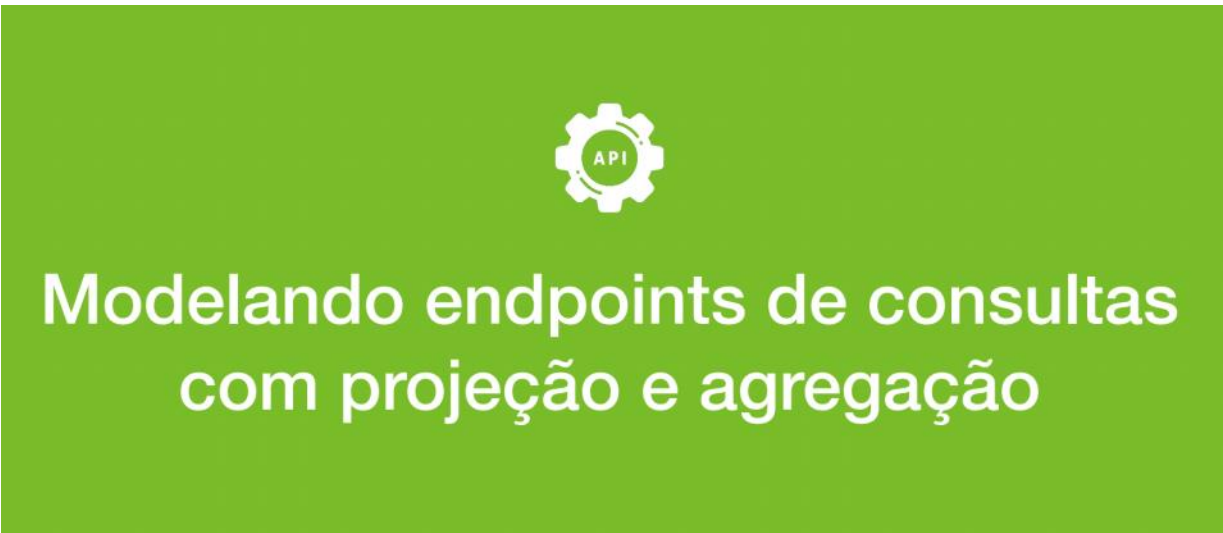
a ideia é criar uma lista de orders percorrendo uma lista de sorts declarados no parâmetro da requisição. Para cada sort percorrido, criar um sort equivalente passando o "nome correto" no valor do fieldsMapping com a chave contida no order.getProperty, ou seja:

requisição:
sort - nomeCliente
-> fieldsMapping(K,V) -> fieldsMapping(nomeCliente, cliente.nome)

o uso do filter é para excluir do stream as chaves que não existem na entidade, evitando o erro de propriedade nula ou vazia, por não conter uma propriedade com o nome invalido ou com erro.

13.12. Modelando endpoints de consultas com dados agregados (ideal para gráficos e dashboards)

sexta-feira, 31 de março de 2023 23:23



No que diz respeito a relatórios e dashboards com informações agregadas, existem diferentes métodos de implementações, desde a URI de recursos a lógica de negócio.

REQUISIÇÃO

```
GET /estatisticas/vendas-diarias?dataCriacaoInicio=2019-10-30T00:00:00Z
&dataCriacaoFim=2019-11-02T23:59:59Z&restauranteId=1
```

RESPOSTA

```
HTTP/1.1 200 OK

[
  {
    "data": "2019-10-30",
    "totalVendas": 1,
    "totalFaturado": 120.00
  },
  {
    "data": "2019-11-02",
    "totalVendas": 2,
    "totalFaturado": 276.60
  }
]
```

Um exemplo de URI de um recurso para obter informações de vendas diárias a fim de criar um relatório ou dashboard de vendas do dia.

- Algumas alternativas de URIs:**
- /restaurantes/1/estatisticas/vendas-diarias
 - /pedidos/estatisticas/vendas-diarias
 - /relatorios/vendas-diarias
 - /insights/vendas-diarias

13.13. Discutindo sobre onde implementar as consultas com dados agregados

sexta-feira, 31 de março de 2023 23:33

No que foi discutido na aula [13.12. Modelando endpoints de consultas com dados agregados \(ideal para gráficos e dashboards\)](#) sobre agregação de dados de diferentes entidades em um recurso, a aula atual conterà implementação baseada na primeira abordagem

- Implementar um subrecurso de coleção recebendo alguns parâmetros de requisição e retorno um array de objetos agregados por dia de acordo com os parâmetros.

GET /estatisticas/vendas-diarias?dataCriacaoInicio=2019-10-30T00:00:00Z&dataCriacaoFim=2019-11-02T23:59:59Z&restauranteId=1

```
[
  {
    "data": "2019-10-30",
    "totalVendas": 1,
    "totalFaturado": 120.00
  },
  {
    "data": "2019-11-02",
    "totalVendas": 2,
    "totalFaturado": 276.60
  }
]
```

- Criar classe que representa vendas diárias
 - Há toda uma discussão por trás da implementação da representação de dados agregados de diferentes entidades, não há um consenso definido pela comunidade, e o ideal é pensar subjetivamente o teor dos dados da informação que será gerada. Um relatório somente de Pedidos é conciso estar dentro do repositório de pedidos, porém, caso contenha outras informações de outras entidades, é uma boa prática criar outra estrutura de repositório.
 - A classe representa os objetos alinhados do array acima

```
package com.algaworks.algafood.domain.model.dto;

import ...

6 usages
@Setter
@Getter
public class VendaDiaria {

    private LocalDate data;
    private Long totalVendas;
    private BigDecimal totalFaturado;
}
```

- Criar um filtro de pesquisa como em [13.6. Implementando pesquisas complexas na API](#)

```
package com.algaworks.algafood.domain.filter;

import ...

6 usages
@Setter
@Getter
public class VendaDiariaFilter {

    private Long restauranteId;
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    private OffsetDateTime dataCriacaoInicio;

    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    private OffsetDateTime dataCriacaoFim;
}
```

- Criar interface de domínio para implementação das regras de filtragem de consultas

```
public interface VendaQueryService {

    1 usage 1 implementation
    List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter);
}
```

- Criar classe que implementa a interface de filtragem de consultas

```
package com.algaworks.algafood.infrastructure.repository.service;

import ...

public class VendaQueryServiceImpl implements VendaQueryService {

    1 usage
    @Override
    public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter) {
        return null;
    }
}
```

- Criar controlador de requisições de consultas

```
@RestController
@RequestMapping("/estatisticas")
public class EstatisticasController {

    1 usage
    @Autowired
    private VendaQueryService vendaQueryService;

    @GetMapping("/vendas-diarias")
    public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter){
        return vendaQueryService.consultarVendasDiarias(vendaDiariaFilter);
    }
}
```

13.14. Implementando consulta com dados agregados de vendas diárias

sábado, 1 de abril de 2023 08:58

- Estrutura da query SQL

```
select date(p.data_criacao) as data_criacao,  
count(p.id) as total_vendas,  
sum(p.valor_total) as total_faturado
```

```
from pedido p
```

```
group by date(p.data_criacao);
```

data_criacao	total_vendas	total_faturado
2023-04-01	1	308.90
2019-09-02	1	79.00
2019-10-30	1	120.00
2019-11-02	2	276.60

- Implementar consulta utilizando Criteria API

```
1 package com.algaworks.algafood.infrastructure.repository.service;  
2  
3 import ...  
14  
15 @Repository  
16 public class VendaQueryServiceImpl implements VendaQueryService {  
17  
18     2 usages  
19     @PersistenceContext  
20     private EntityManager entityManager;  
21  
22     1 usage  
23     @Override  
24     public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter) {...}  
40  
41 }
```

- 1º passo: criar os elementos para a criação da consulta com CRITERIA API
 - Contexto de persistência com @PersistenceContext
 - Objeto CriteriaBuilder a partir do contexto de persistência
 - Objeto CriteriaQuery a partir de um CriteriaBuilder chamando createQuery com o argumento da classe resultante do retorno, ou seja, VendaDiaria.
 - Objeto Root a partir de um CriteriaQuery chamando o método from passando no argumento a entidade de consulta, no caso, Entidade(coluna) Pedido.

```
final CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();  
final CriteriaQuery<VendaDiaria> query = criteriaBuilder.createQuery(VendaDiaria.class);  
final Root<Pedido> root = query.from(Pedido.class);
```

- 2º passo: construir a query usando os elementos da Criteria criados acima
 - chamar o método construct do objeto criteriaBuilder passando no 1º argumento a classe que deseja construir utilizando o retorno da query, por

isso é necessário o construtor da classe com todos os atributos. Cada registro retornado da query SQL no banco será convertido em nesse objeto, no caso VendaDiaria.

- O 2º argumento será criado posteriormente, pois é a função nativa do Mysql para converter uma data com tempo e somente Data.
date(p.data_criacao)
- O 3º argumento é a contagem do total de vendas por dia, chamando o método count do objeto criteriaBuilder passando no argumento o root (pedido) usando get("id") não precisa ser necessariamente o id, é somente para fins de contagens, poderia ser a contagem de tudo como no SQL count(*)
- o 4º argumento é o método sum do objeto criteriaBuilder, chamando o root usando o get passando o nome da propriedade desejada para fazer o somatório, no caso root.get("valorTotal")

```
final CompoundSelection<VendaDiaria> selection = criteriaBuilder.construct(VendaDiaria.class,  
    ...selections: null,  
    criteriaBuilder.count(root.get("id")),  
    criteriaBuilder.sum(root.get("valorTotal")));
```

- Construção da função date para o 2º argumento do construct da query
 - chamar o método function do objeto criteriaBuilder, o método aceita no 1º argumento o nome da função nativa, no caso, 'date', o 2º argumento aceita o tipo de objeto que será convertido após a execução da função, e depois um varargs de expressões (root.get), no caso vamos chamar root.get("dataCriacao") para adicionar no argumento da função. Será retornado uma expressão que vai ser adicionado no 2º argumento do passo 2.

```
final Expression<Date> function = criteriaBuilder.function( name: "date", Date.class, root.get("dataCriacao"));  
  
final CompoundSelection<VendaDiaria> selection = criteriaBuilder.construct(VendaDiaria.class,  
    ...selections: function,  
    criteriaBuilder.count(root.get("id")),  
    criteriaBuilder.sum(root.get("valorTotal")));
```

- Após a construção da query, guardar em um objeto Selection para ser usado na projeção.

```
final CompoundSelection<VendaDiaria> selection = criteriaBuilder.construct(VendaDiaria.class,  
    ...selections: function,  
    criteriaBuilder.count(root.get("id")),  
    criteriaBuilder.sum(root.get("valorTotal")));
```

Nota: no 2º passo, é necessário que as expressões criadas estão na mesma ordem que os atributos do construtor da classe de construção (1º argumento)

```
@AllArgsConstructor  
@Setter  
@Getter  
public class VendaDiaria {  
  
    private Date data;  
    private Long totalVendas;  
    private BigDecimal totalFaturado;  
}
```

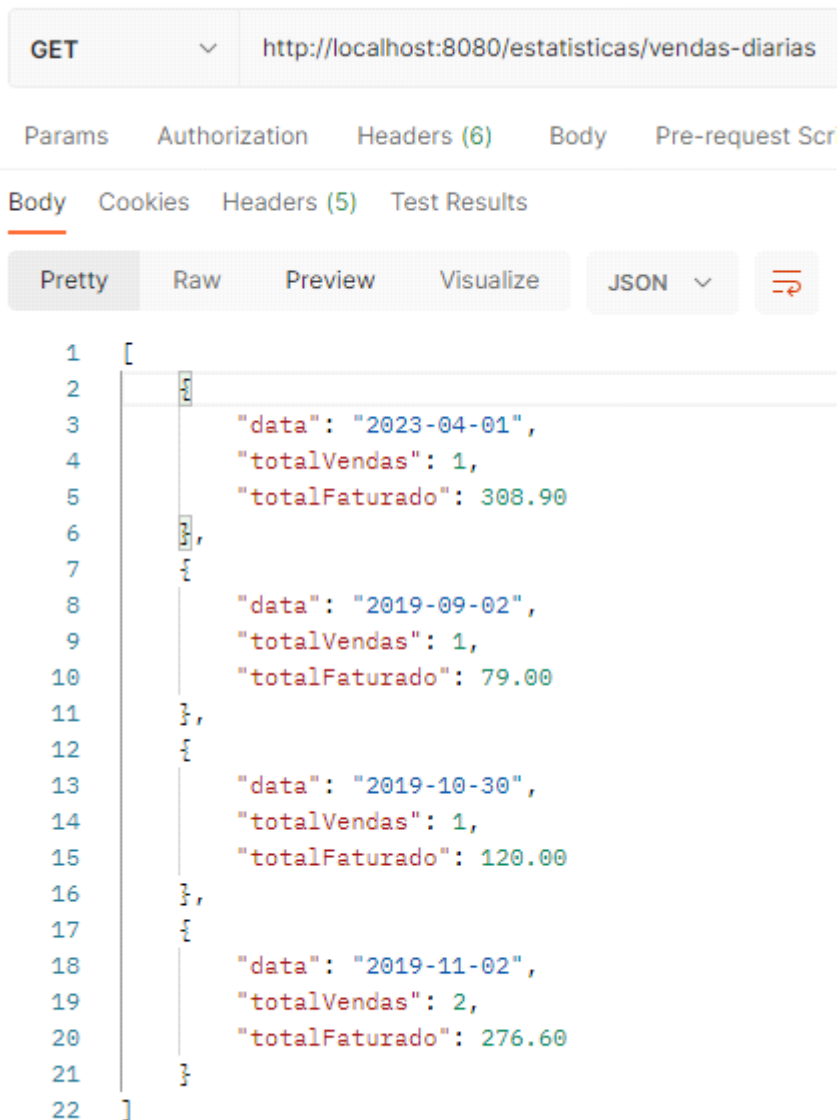
- 3º passo: projetar a query
 - chamar o método select do objeto criteriaQuery passando a selection criada, sem retorno.

- agrupar pela data convertida da função date criada acima, chamando o `groupBy` do objeto `criteriaQuery` passando a função `date`.
- retornar chamando o método `createQuery` do `entityManager` passando o objeto `criteriaQuery`.

```
final CriteriaQuery<VendaDiaria> select = query.select(selection);
query.groupBy( ...grouping: function);

return entityManager.createQuery(query).getResultList();
```

Chamada da requisição



The screenshot shows a REST client interface with a GET request to `http://localhost:8080/estatisticas/vendas-diarias`. The response is displayed in JSON format, showing a list of 4 objects. Each object contains the date, total number of sales, and total revenue.

data	totalVendas	totalFaturado
"2023-04-01"	1	308.90
"2019-09-02"	1	79.00
"2019-10-30"	1	120.00
"2019-11-02"	2	276.60

```

@Repository
public class VendaQueryServiceImpl implements VendaQueryService {

    2 usages
    @PersistenceContext
    private EntityManager entityManager;

    1 usage
    @Override
    public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter){

        final CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
        final CriteriaQuery<VendaDiaria> query = criteriaBuilder.createQuery(VendaDiaria.class);
        final Root<Pedido> root = query.from(Pedido.class);

        final Expression<Date> function = criteriaBuilder.function( name: "date", Date.class, root.get("dataCriacao"));

        final CompoundSelection<VendaDiaria> selection = criteriaBuilder.construct(VendaDiaria.class,
            ...selections: function,
            criteriaBuilder.count(root.get("id")),
            criteriaBuilder.sum(root.get("valorTotal")));

        final CriteriaQuery<VendaDiaria> select = query.select(selection);
        query.groupBy( ...grouping: function);

        return entityManager.createQuery(query).getResultList();
    }
}

```

13.16. Tratando time offset na agregação de vendas diárias por data

sábado, 1 de abril de 2023 17:57

No banco de dados, as datas estão formatadas no padrão UTC (+00:00) e podem acabar sendo projetadas em horários diferentes

```
    , data_confirmacao, data_e
efe58dc2', 1, 2, 1, 1, '38
', '2019-11-02 20:35:10',
produto_id, quantidade, pr

nte_id, usuario_cliente_id
endereço_numero, endereço
, data_confirmacao, data_e
38754d63', 1, 3, 2, 1, '38
', '2019-11-03 02:01:21',
```

No horário em UTC, os dois pedidos foram em dias diferentes

	subtotal	valor_total	restaurante_id	data_criacao	status
1	298.90	308.90	1	2023-04-01 21:32:46	CRIADO
2	79.00	79.00	4	2019-09-02 21:00:30	CRIADO
3	110.00	120.00	1	2019-10-30 21:10:00	ENTREGUE
4	174.40	179.40	1	2019-11-02 20:34:04	ENTREGUE
5	87.20	97.20	1	2019-11-03 02:00:30	ENTREGUE

Mas no horário de Brasília (+03:00) temos 3 horas a menos, ou seja, ambos os pedidos foram no mesmo dia. Porém, o banco de dados não trata isso.

	data_criacao	total_vendas	total_faturado
1	2019-10-30	1	120.00
2	2019-11-02	1	179.40
3	2019-11-03	1	97.20

** O pedido de id 2 e 3 são no mesmo dia, apenas 3 horas de diferença

Uma forma é adicionando uma função do Mysql na projeção que trata as 3 horas de diferença

```
convert_tz(p.data_criacao,'+00:00','-03:00')

select date(convert_tz(p.data_criacao, '+00:00', '-03:00')) as data_criacao,
count(p.id) as total_vendas,
sum(p.valor_total) as total_faturado

from pedido p

where p.status in ('CONFIRMADO', 'ENTREGUE')

group by date(convert_tz(p.data_criacao, '+00:00', '-03:00'));
```

	data_criacao	total_vendas	total_faturado
1	2019-10-30	1	120.00
2	2019-11-02	2	276.60

Agora é necessário implementar uma função na CriteriaApi para projetar

```
final Expression<Date> convert_tz = criteriaBuilder.
    function( name: "convert_tz", Date.class, root.get("dataCriacao"),
              criteriaBuilder.literal( value: "+00:00"), criteriaBuilder.literal(timeOffSet));
```

A função convert_tz equivalente ao MYSQL nativo possui 5 argumentos, o nome da função correspondente, o tipo de retorno, o atributo da entidade a ser selecionado, o offset padrão e o novo offset de acordo com a requisição. o último argumento pede uma instância de Expression, logo, tempos que chamar o método literal do criteriaBuilder

```
private VendaQueryService vendaQueryService;

@GetMapping("/vendas-diarias")
public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter,
    @RequestParam (required = false, defaultValue = "+00:00") String timeOffSet){
    return vendaQueryService.consultarVendasDiarias(vendaDiariaFilter, timeOffSet);
}
```

13.17. Conhecendo o JasperSoft Studio

sábado, 1 de abril de 2023 19:30

- Biblioteca de criação de relatório em diferentes formatos

13.18. Criando um layout do relatório JasperReports de vendas diárias

sábado, 1 de abril de 2023 19:58

13.19. Estruturando endpoint e serviço de emissão de relatório em PDF

sábado, 1 de abril de 2023 21:18

- Utilizar a mesma URL para gerar relatório
 - Gerar Content Negotiation com Json e Pdf

```
@GetMapping(value = "/vendas-diarias", produces = MediaType.APPLICATION_JSON_VALUE)
public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter,
    @RequestParam (required = false, defaultValue = "+00:00") String timeOffSet){
    return vendaQueryService.consultarVendasDiarias(vendaDiariaFilter, timeOffSet);
}

@GetMapping(value = "/vendas-diarias", produces = MediaType.APPLICATION_PDF_VALUE)
public List<VendaDiaria> consultarVendasDiariasPdf(VendaDiariaFilter vendaDiariaFilter,
    @RequestParam (required = false, defaultValue = "+00:00") String timeOffSet){
    return vendaQueryService.consultarVendasDiarias(vendaDiariaFilter, timeOffSet);
}
```

Também é aceitável substituir o nome do atributo da anotação de 'value' para 'path'

```
@GetMapping(path = "/vendas-diarias", produces = MediaType.APPLICATION_JSON_VALUE)
public List<VendaDiaria> consultarVendasDiarias(VendaDiariaFilter vendaDiariaFilter,
    @RequestParam (required = false, defaultValue = "+00:00") String timeOffSet){
    return vendaQueryService.consultarVendasDiarias(vendaDiariaFilter, timeOffSet);
}

@GetMapping(path = "/vendas-diarias", produces = MediaType.APPLICATION_PDF_VALUE)
public List<VendaDiaria> consultarVendasDiariasPdf(VendaDiariaFilter vendaDiariaFilter,
    @RequestParam (required = false, defaultValue = "+00:00") String timeOffSet){
    return vendaQueryService.consultarVendasDiarias(vendaDiariaFilter, timeOffSet);
}
```

```
@GetMapping(path = "/vendas-diarias", produces = MediaType.APPLICATION_PDF_VALUE)
public ResponseEntity<byte[]> consultarVendasDiariasPdf(VendaDiariaFilter vendaDiariaFilter,
    @RequestParam (required = false, defaultValue = "+00:00") String timeOffSet){
    final byte[] bytesPdf = vendaReportService.emitirVendasDiarias(vendaDiariaFilter, timeOffSet);

    final HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.CONTENT_DISPOSITION, headerValue: "attachment; filename=vendas-diarias.pdf");

    return ResponseEntity.ok().headers(headers).contentType(MediaType.APPLICATION_PDF).body(bytesPdf);
}
```

O campo de headers é configurado para a requisição retornar um arquivo para download

a interface VendaReportService

```
package com.algaworks.algafood.domain.service;

import com.algaworks.algafood.domain.filter.VendaDiariaFilter;

4 usages 1 implementation
public interface VendaReportService {

    1 usage 1 implementation
    byte[] emitirVendasDiarias(VendaDiariaFilter filtro, String timeOffSet);
}
```

A implementação da interface, note que, está anotada com @Service, porém, dentro do pacote de repositórios, pois está próximo a instruções no banco de dados.

```
package com.algaworks.algafood.infrastructure.repository.service.report;

import ...

@Service
public class VendaPdfReportServiceImpl implements VendaReportService {

    1 usage
    @Override
    public byte[] emitirVendasDiarias(VendaDiariaFilter filtro, String timeOffSet) { return null; }
}
```

<input checked="" type="checkbox"/>	Accept-Encoding	①	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection	①	keep-alive
<input checked="" type="checkbox"/>	Accept		application/pdf, application/json
<input type="checkbox"/>	Accept		application/json
	Key		Value

Quando a requisição contém valores inválidos, o erro é tratado pelo ExceptionHandler e retorna um Json, mas quando a requisição é configurada para aceitar pdf, ocorre um erro diferente do esperado, a requisição é tratada mas em seguida ocorre o erro de tentar mostrar Json. Então é necessário adicionar um segundo valor para 'Accept' para também aceitar Json no lugar de Pdf.

13.20. Preenchendo um relatório JasperReports com JavaBeans e gerando bytes do PDF

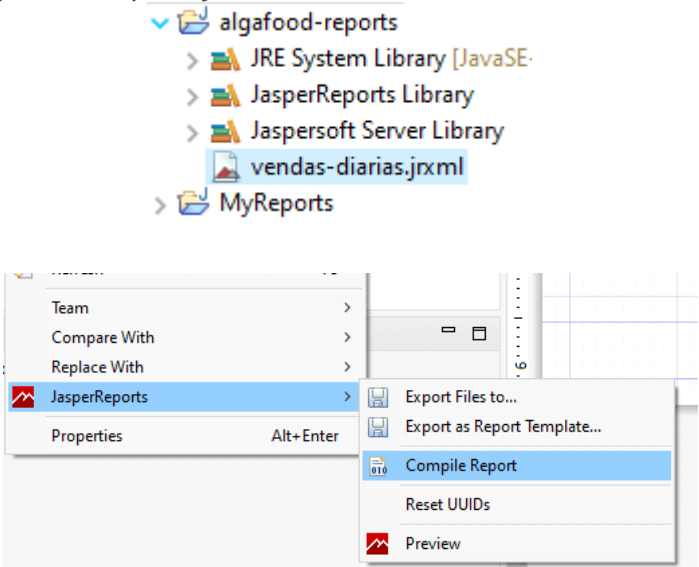
domingo, 2 de abril de 2023 07:55

- Importar a api do jasper reports

```
<!--https://mvnrepository.com/artifact/net.sf.jasperreports/jasperreports-->
<dependency>
<groupId>net.sf.jasperreports</groupId>
<artifactId>jasperreports</artifactId>
<version>6.20.1</version>
</dependency>
<!--https://mvnrepository.com/artifact/net.sf.jasperreports/jasperreports-functions-->
<dependency>
<groupId>net.sf.jasperreports</groupId>
<artifactId>jasperreports-functions</artifactId>
<version>6.20.1</version>
</dependency>
```

A depedência do jasperreports-functions é necessária para converter os os atributos dento do jasper reports

- Compilar o arquivo jrxml



```
@Service
public class VendaPdfReportServiceImpl implements VendaReportService {

    1 usage
    @Autowired
    private VendaQueryService vendaQueryService;

    1 usage
    @Override
    public byte[] emitirVendasDiarias(VendaDiariaFilter filtro, String timeOffSet) {

        final InputStream inputStream = this.getClass().getResourceAsStream( name: "/relatorios/vendas-diarias.jasper");

        final HashMap<String, Object> parametros = new HashMap<>();
        parametros.put("REPORT_LOCALE", new Locale( language: "pt",  country: "BR"));

        final List<VendaDiaria> vendaDiarias = vendaQueryService.consultarVendasDiarias(filtro, timeOffSet);
        final JRBeanCollectionDataSource dataSource = new JRBeanCollectionDataSource(vendaDiarias);

        try {
            JasperPrint jasperPrint = JasperFillManager.fillReport(inputStream, parametros, dataSource);
            return JasperExportManager.exportReportToPdf(jasperPrint);
        } catch (Exception e) {
            throw new ReportException(ErrorMessage.RELATARIO_INDISPONIVEL.get(), e);
        }
    }
}
```