12.1. Modelando sub-recursos para relacionamentos

quarta-feira, 15 de março de 2023 22:58

12.2. Granularidade de recursos: Chatty vs Chunky APIs

quarta-feira, 15 de março de 2023 23:00

APIS com granularidade fina: Chatty API - API tagarela pois o consumidor precisa fazer várias chamadas a api para realizar uma operação comum

exemplo: Cadastrar um novo restaurante, depois um PUT para atribuir um endereço, depois um PUT para adicionar uma taxaFrete . Logo, fazemos várias requisições para realizar uma operação, pois os recursos da api tem uma granularidade fina.

Uma Chunky API contém recursos com granularidade grossa, na qual um recurso pode conter vários atributos variados mas concisos para a realização de uma única operação, como por exemplo: um único POST para cadastrar um restaurante e seus detalhes adicionais como endereço, taxa de frente, status de funcionamento, cozinhas relacionadas e produtos.

Chatty vs Chunky

Para analisar a granularidade dos recursos e chegar em uma resposta devemos pensar nos consumidores da API. Eles devem conseguir fazer o que devem fazer.

Pensando em granularidade grossa:

> Se o recurso do Restaurante incluir endereço e suas informações, o consumidor da API deverá conhecer o endereço e tratar o endereço

Pensando em granularidade fina:

> Se o recurso conter subrecursos para tratar informações relacionadas ao recurso principal, o consumidor pode consultar e atualizar esses subrecursos como endereço do restaurante de forma independente, além de consultar e atualizar dados do restaurante sem tratar de endereços.

Há a ressalva ao modelar recursos com granularidade fina para não deixar o recurso inconsistente, como por exemplo: adicionar uma entidade em um recurso que contém atributos não nulos em um outro subrecurso, adicionando acidentalmente ou não, uma entidade coma atribulo não nulo, no caso de adicionar um restaurante em um recurso e não adicionar um endereco em um outro subrecurso.

Outro caso em questão é a sequência correta de adição de recursos para realizar uma única operação no lado do consumidor da API. Por exemplo: A regra de negócio diz que um endereço é obrigatório ao cadastrar um restaurante e para adicionar um produto a um restaurante, esse restaurante terá que conter um endereço; a sequência fica: cadastrar um endereço, cadastrar um restaurante com o endereço cadastrado e depois cadastrar produtos. A regra de negócio relacionada a sequência de cadastross possui possibilidade de inconsistência mais pro lado do consumidor, seguindo à risca a sequência de chamadas a api.

Se existir a possibilidade de deixar o recurso em um estado inconsistente em uma granularidade fina, é mais confiável ter o recurso com granularidade grossa. Caso o consumidor precise fazer coisas específicas na API que não fazem sentido conter em um recurso com granularidade grossa, desde que não deixe o recurso inconsistente, é melhor deixar com granularidade fina.

12.3. Modelando conceitos abstratos de negócio e ações não-CRUD como recursos

quinta-feira, 16 de março de 2023 00:13

Podemos ter métodos que não se encaixam em um CRUD, na aula iremos discutir alternativas e melhores práticas nesses cenários.

recurso com funcionalidade de ativação e inativação e não necessariamente pode existir na entidade do domain model ou do banco de dados. Temos um recurso com granularidade grossa.

Dependendo da funcionalidade desencadeada ao ativar ou inativar um recurso através da propriedade no payload da requisição, a modelagem da ativação/inativação poderá deixar o código mais complexo.

Podemos alternar a requisição para PATCH e passar no payload somente o campo de ativação/inativação, mas esbarramos em possíveis regras de negócios a serem validadas antes de executar a ativação/inativação.

Uma alternativa viável também é criar sub-recursos da entidade, criando endpoints somente para tal finalidade.

Quando nomeamos recursos, utilizamos substantivos.

Podemos extrair o conceito abstrato de negócio da entidade e modelarmos como recursos. Não necessariamente deve existir entidades para os recursos.

Ou criar apenas um subrecurso com 1 endpoint passando no payload da requisição um valor booleano.

Ou criar endpoint com verbos HTTP diferentes para a ativação ou inativação.

No lugar de extrair a ativação e inativação, podemos voltar a pensar em tornar concreto o conceito abstrato do negócio ativação/inativação. Ou seja, há um conceito abstrato de negócio da entidade que ativa e inativa, podemos extrair e tornar concreto o conceito esse conceito ao criar um subrecurso do recurso que contém a ativação/inativação.

Podemos fazer um POST em um subrecurso de alteração de status passando no corpo da requisição um valor booleano, a vantagem é armazenar informações do subrecurso a fim de, com GET, obter o histórico com detalhes das alterações da entidade. Mesmo que não armazenemos as informações, é viável fazer ainda dessa forma.

Um exemplo de pagamento de uma compra no qual não existe uma entidade Pagamento, ou seja, estamos abstraindo o conceito de pagamento, pois, para que uma compra seja finalizada, é necessário um pagamento atribuído à compra. As propriedades do payload do sub-recurso não necessariamente devem conter na entidade de domínio do recurso.

Um exemplo hipotético seria criar um recurso para enviar notificações a todos os restaurantes cadastrados na aplicação, não temos uma entidade notificação mas podemos modelar um recurso para disparar notificações aos emails de cadastro dos restaurantes para todos os restaurantes.

12.4 Implementando os endpoints de ativação e inativação de restaurantes

quinta-feira, 16 de março de 2023 21:07

```sql
alter table restaurante add ativo boolean not null;
update restaurante set ativo = 1;
```

- testes de sql a fim de validar a execução da query no banco de dados
- Criar uma nova migração
- alterar ddls do after migrate em tabelas restaurantes adicionando a nova coluna 'ativo'
- Adicionar atributos na entidade e DTO
- Implementar regras de negócio de ativação e inativação na service
- Alternativas de implementação de ativação/inativação da entidade

```java
@Transactional
public void ativar(Long restauranteId){
    final Restaurante restaurante = buscarOuFalhar(restauranteId);
    restaurante.setAtivo(Boolean.TRUE);
}

@Transactional
public void inativar(Long restauranteId){
    final Restaurante restaurante = buscarOuFalhar(restauranteId);
    restaurante.setAtivo(Boolean.FALSE);
}
```

ou

```java
@Transactional
public void ativar(Long restauranteId){
    final Restaurante restaurante = buscarOuFalhar(restauranteId);
    restaurante.ativar();
}

@Transactional
public void inativar(Long restauranteId){
    final Restaurante restaurante = buscarOuFalhar(restauranteId);
    restaurante.inativar();
}
```

- Implementar métodos de ativação e inativação no controller

```java
@PutMapping("/{restauranteId}/ativar")
public ResponseEntity<Void> ativar(@PathVariable Long restauranteId){
    restauranteService.ativar(restauranteId);
    return ResponseEntity.noContent().build();
}

@DeleteMapping("/{restauranteId}/inativar")
public ResponseEntity<Void> inativar(@PathVariable Long restauranteId){
    restauranteService.inativar(restauranteId);
    return ResponseEntity.noContent().build();
}
```

Nota: estamos abstraindo o conceito de ativação/inativação de uma entidade e criando um sub-recurso a partir de um atributo.

12.5. Desafio- implementando os endpoints de formas de pagamento

sexta-feira, 17 de março de 2023 23:00

- CRUD da entidade FormaPagamento
- Utilizar DTO
- Utilizar Input DTO
- Criar exceções personalizadas
- Validar atributos com Bean Validation
- Personalizar mensagens do Bean Validation
- Criar Controller
- Criar Service
- Criar Repository

12.6. Adicionando endereço no modelo da representação do recurso de restaurante

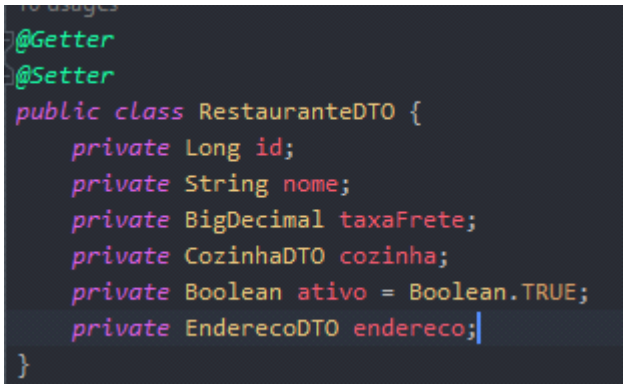
sábado, 18 de março de 2023 20:50

Até o momento não estávamos adicionando o atributo Endereço de Restaurante na representação do recurso. Na aula vamos adicionar na representação GET.

- Criar uma nova classe para ser o modelo de representação do endereço de Restaurante
- Dica: em um DTO quando há referência para outra entidade, utilizar o DTO da entidade em questão, caso não houver, criar um DTO com o atributo requerido.

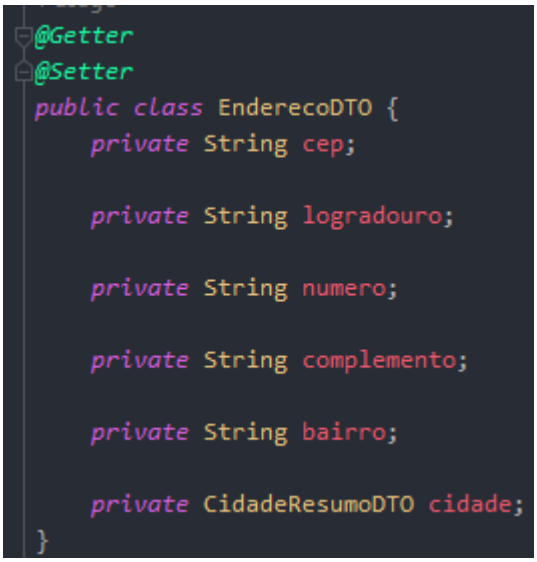
  Para mudar a representação de um DTO específico sem alterar a representação do mesmo em outras chamadas, basta criar um DTO específico. A criação de DTOS para a representação não tem regras, mas há de se utilizar com bom senso.

  Restaurante



```
@Getter
@Setter
public class RestauranteDTO {
    private Long id;
    private String nome;
    private BigDecimal taxaFrete;
    private CozinhaDTO cozinha;
    private Boolean ativo = Boolean.TRUE;
    private EnderecoDTO endereco;
}
```

  EnderecoDTO

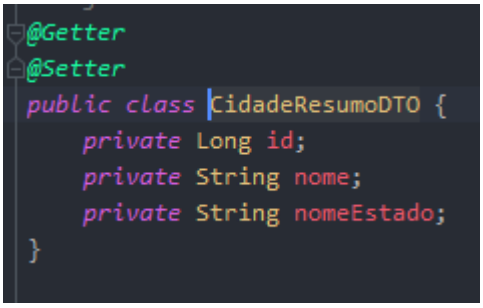


```
@Getter
@Setter
public class EnderecoDTO {
    private String cep;

    private String logradouro;

    private String numero;

    private String complemento;

    private String bairro;

    private CidadeResumoDTO cidade;
}
```

  A representação do recurso estava complexo demais com objetos alinhados, então como alternativa, dentro da representação de "cidade" tiramos o objeto Estado e adicionamos à uma String: cidade:{nome, nomeEstado}

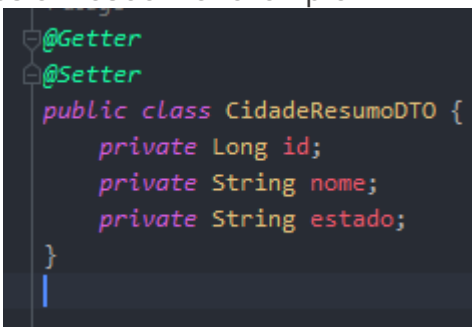
```
{
    "id": 1,
    "nome": "Thai Gourmet",
    "taxaFrete": 10.00,
    "cozinha": {
        "id": 1,
        "nome": "Tailandesa"
    },
    "ativo": true,
    "endereco": {
        "cep": "38400-999",
        "logradouro": "Rua João Pinheiro",
        "numero": "1000",
        "complemento": null,
        "bairro": "Centro",
        "cidade": {
            "id": 1,
            "nome": "Uberlândia",
            "nomeEstado": "Minas Gerais"
        }
    }
}
```


Foi necessário criar a classe DTO CidadeResumoDTO para não alterar a classe CidadeDTO que era usada em outras representações de recurso. A ideia é criar uma classe para representar como a imagem acima.



O modelMapper é capaz de adicionar o nome do Estado a String e fazer o cast

O modelMapper não é capaz de adicionar na String o nome do estado quando o atributo é o mesmo da entidade. Por exemplo:



```
"endereco": {
    "cep": "38400-999",
    "logradouro": "Rua João Pinheiro",
    "numero": "1000",
    "complemento": null,
    "bairro": "Centro",
    "cidade": {
        "id": 1,
        "nome": "Uberlândia",
        "estado": "Estado(id=1, nome=Minas Gerais)"
    }
}
```


Pois ele captura a String do toString da entidade Estado. Para termos o mesmo efeito quando colocamos o atributo "nomeEstado" temos que fazer uma configuração manual do mapeamento na definição do Bean.

- Configuração manual do mapeamento do ModelMapper para customizar o valor de um atributo

```java
@Configuration
public class ModelMapperConfig {

    @Bean
    public ModelMapper modelMapper(){

        ModelMapper modelMapper = new ModelMapper();
        //11.16. Customizando o mapeamento de propriedades com ModelMapper
        ///  modelMapper.createTypeMap(Restaurante.class, RestauranteDTO.class)
        //          .addMapping(Restaurante::getTaxaFrete, RestauranteDTO::setPrecoFrete);

        TypeMap<Endereco, EnderecoDTO> typeMap = modelMapper.createTypeMap(Endereco.class, EnderecoDTO.class);
        typeMap.<String>addMapping(src -> src.getCidade().getEstado().getNome(),
                (dest, value) -> dest.getCidade().setEstado(value));

        return modelMapper;
    }
}
```

Retorno JSON:

```
{
    "id": 1,
    "nome": "Thai Gourmet",
    "taxaFrete": 10.00,
    "cozinha": {
        "id": 1,
        "nome": "Tailandesa"
    },
    "ativo": true,
    "endereco": {
        "cep": "38400-999",
        "logradouro": "Rua João Pinheiro",
        "numero": "1000",
        "complemento": null,
        "bairro": "Centro",
        "cidade": {
            "id": 1,
            "nome": "Uberlândia",
            "estado": "Minas Gerais"
        }
    }
}
```


na linha 21 criamos o tipo de mapeamento entre as classes. A origem dos atributos vem da entidade Endereco.class (incluindo cidade e estado) e queremos atribuir no EnderecoDTO para representar o recurso na resposta.

Fazendo da forma normal, temos a String do toString de estado dentro de Endereco.class


```
"endereco": {
    "cep": "38400-999",
    "logradouro": "Rua João Pinheiro",
    "numero": "1000",
    "complemento": null,
    "bairro": "Centro",
    "cidade": {
        "id": 1,
        "nome": "Uberlândia",
        "estado": "Estado(id=1, nome=Minas Gerais)"
    }
}
```


na linha 22 adicionamos o mapeamento. Usamos o addMapping com o método que recebe 2 interfaces funcionais como argumento, o 1º método recebe a origem do método da classe de origem, no caso de Endereco, e o 2º argumento recebe uma interface funcional que recebe 2 argumentos, o 1º é a classe de destino, no caso EnderecoDTO, e o segundo é o valor, esse valor veio do 1º argumento do addMapping, no corpo da expressão lamba chamamos via get e setamos o nome do estado no atributo.

```
    "endereco": {
        "cep": "38400-999",
        "logradouro": "Rua João Pinheiro",
        "numero": "1000",
        "complemento": null,
        "bairro": "Centro",
        "cidade": {
            "id": 1,
            "nome": "Uberlândia",
            "estado": "Minas Gerais"
        }
    }
}
```

12.7. Refatorando serviço de cadastro de restaurante para incluir endereço

sábado, 18 de março de 2023 23:57

Incluir no corpo da requisição, um objeto endereço para ser adicionado em um POST.
Obrigando o consumidor da API a enviar um Endereço.
Não devemos mudar no banco de dados as colunas que não são not null, pois iria quebrar o banco e a aplicação.

- Criar classe DTO que recebe na requisição via POST ou PUT (input)
- Criar classe de referência para cidade CidadeRef
- Para tratar várias exceções de um mesmo tipo, podemos separar as exceções com |

```java
@PutMapping("/{id}")
public RestauranteDTO atualizar(@RequestBody @Valid RestauranteInputDTO restauranteInput, @PathVariable Long id) {
    final Restaurante restauranteAtual;

    try {
        Cozinha cozinha = cozinhaService.buscar(restauranteInput.getCozinha().getId());
        Cidade cidade = cidadeService.buscar(restauranteInput.getEndereco().getCidade().getId());

        restauranteAtual = restauranteService.buscar(id);
        rInputDisassembler.copyProperties(restauranteInput, restauranteAtual);

        restauranteAtual.setCozinha(cozinha);
        restauranteAtual.getEndereco().setCidade(cidade);

        return rAssembler.toDTO(restauranteService.atualizar(restauranteAtual));

    } catch (CozinhaNaoEncontradaException | CidadeNaoEncontradaException e) {
        throw new NegocioException(e.getMessage(), e);
    }
```

12.8. Desafio- implementando os endpoints de grupos

domingo, 19 de março de 2023 11:05

- Criar DTO e Input DTO da entidade e mapear com anotações do Bean Validation
- Criar classe assembler para o ModelMapper para conversão de entidades
- Criar exceção para entidade não encontrada
- Criar repository, service e controller
- Adicionar mensagens customizadas para validações do Bean Validation
- Adicionar query sql para massa de dados

12.9. Desafio: implementando os endpoints de usuarios

domingo, 19 de março de 2023 12:27

- Criar DTO e Input DTO da entidade e mapear com anotações do Bean Validation
- Ciar DTO para alteração exclusiva de senha
- Implementar endpoint específico para alteração de senha
- Criar classe assembler para o ModelMapper para conversão de entidades
- Criar exceção para entidade não encontrada
- Criar repository, service e controller
- Adicionar mensagens customizadas para validações do Bean Validation
- Adicionar query sql para massa de dados

12.10. Um pouco mais sobre JPA: objeto alterado fora da transação é sincronizado com o banco de dados

domingo, 19 de março de 2023 15:42

Quando um objeto gerenciado pelo contexto de persistência do Entity Manager é alterado, os efeitos refletem no banco de dados ao passar por uma transação. Ou seja, mesmo antes de chamarmos um método para salvar um objeto, como por exemplo, EntityManager.save(Entity s) ou entidadeRepository.save(Entity S), ele reflete a alteração.

Isso se dá pelo estado da instância gerenciada, que ao passar por um método anotado com @Transactional, entra em um período de transação com o banco de dados.

```
@PutMapping("/{usuarioId}")
public UsuarioDTO atualizar(@RequestBody @Valid usuarioInputUpdateDTO usua

    Usuario usuarioAtual = usuarioService.buscar(usuarioId);
    aAssembler.copyProperties(usuarioUpdateDTO, usuarioAtual);

    return aAssembler.toDTO(usuarioService.atualizar(usuarioAtual));
}
```

O objeto gerenciado usuarioAtual sofre alteração na chamada do método copyProperties, e ao entrar no método atualizar que está anotado com @Transactional, sofre alterações no banco de dados.

```
@Transactional
public Usuario atualizar(Usuario usuario) {

    return usuarioRepository.save(usuario);
}
```

Mesmo não contento a chamada para save em usuarioRepository o objeto sofreria alterações no banco de dados ao entrar no método anotado com @Transactional, pois o começo do método equivale a chamada .begin e já reflete no banco de dados e o término do método equivale ao .commit, ambos do entityManager. Caso aconteça um lançamento de exceção durante o método, o Spring Data JPA fará um rollback, pois o método não foi concluído. Isso acontece com métodos anotados com @Transacional.

```
35      @Transactional
36      public Usuario atualizar(Usuario usuario) {
37          final Usuario save = usuarioRepository.save(usuario);
38
39          if(true)
40              throw new RuntimeException();
41          return save ;
42      }
```

12.11. Implementando regra de negócio para evitar usuários com e-mails duplicados

domingo, 19 de março de 2023 17:06

```java
@Transactional
public Usuario salvar(Usuario usuario) {
    usuarioRepository.detach(usuario); /*Antes de chamaro findbyEmail o SDJPA faz o commit dos objetos gerenciados
    por isso é necessário desanexar do contexto de persistencia, pois vai adicionar no bd um usuari com o mesmo
    email*/

    final Optional<Usuario> usuarioExistente = usuarioRepository.findByEmail(usuario.getEmail());

    /*Se um usuario vindo do banco com o mesmo email for diferente do usuario vindo da requisição, cai no if */
    if(usuarioExistente.isPresent() && !usuarioExistente.get().equals(usuario)){
        throw new NegocioException(String.format(ErrorMessage.EMAIL_JA_CADASTRADO.get(), usuario.getEmail()));
    }
    return usuarioRepository.save(usuario);
}
```

Primeiro adicionamos um método na interface do repositório

```java
@Repository
public interface UsuarioRepository extends CustomJpaRepository<Usuario, Long> {

    1 usage
    Optional<Usuario> findByEmail(String email);
}
```

Para trazer usuários com o e-mail passado no argumento.

Verificamos se há um usuário no Optional EE se o usuário vindo do banco de dados é o mesmo usuário que está vindo da requisição, pois o usuário pode estar atualizando outro dado além do e-mail.

```java
32      usuarioRepository.detach(usuario); /*Antes de chamaro findbyEmail o SDJPA faz o commit dos objetos gerenciados
33      por isso é necessário desanexar do contexto de persistencia, pois vai adicionar no bd um usuari com o mesmo
34      email*/
```

na linha 32 está desanexando uma entidade do contexto de persistência, pois a entidade usuario é gerenciada pelo contexto de persistência, e ao atualizar a entidade (que tem o mesmo email que o do registro no BD), é adicionada no banco de dados antes de chamar .findByEmail, para manter a integridade dos registros concisos. Logo, a chamada em findByEmail traz dois registros, o registro vindo da requisição e o registro já contido no banco, e ambos são a mesma entidade, claro, com id's diferente por conta da persistência no banco antes da chamada. Por isso o uso do detach para não gerenciar mais a entidade.

O detach pode ser adicionado em uma nova implementação do repositório, estendendo o SimpleJpaRepository, por exemplo a CustomJpaRepository criada manualmente.

```java
2 usages
public class CustomJpaRepositoryImpl<T, ID>
        extends SimpleJpaRepository<T, ID>
        implements CustomJpaRepository<T, ID> {
```

```java
1 usage
@Override
public void detach(T entity) { entityManager.detach(entity); }
```

e estendemos a CustomJpaRepository no repositório da entidade.

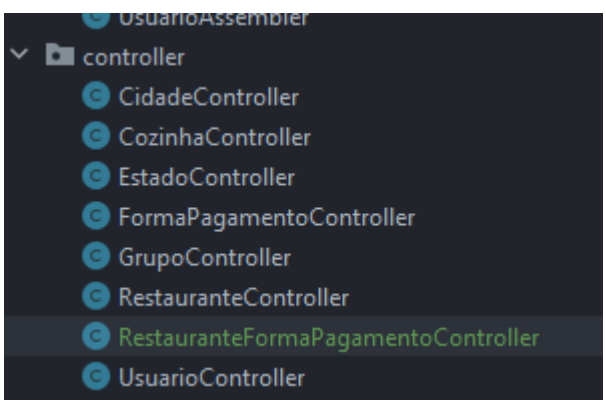
12.12. Implementando os endpoints de associação de formas de pagamento em restaurantes

domingo, 19 de março de 2023 19:39

Implementação de um endpoint de um recurso de uma entidade que possui relacionamento Many to Many

PUT restaurantes/id/formas-pagamentos/id
GET restaurantes/id/formas-pagamento

Foi criada uma classe controladora de formas de pagamento de restaurantes para não ficar poluído o controller principal



Foi adicionado 3 métodos para requisição. Para associação, desassociação e listagem de formas de pagamento vinculados a restaurantes.

```
    @GetMapping
    public List<FormaPagamentoDTO> listar(@PathVariable Long restauranteId) {
        final Restaurante restaurante = restauranteService.buscar(restauranteId);

        return fPAssembler.toListDTO(restaurante.getFormasPagamento());
    }

    @DeleteMapping("/{formaPagamentoId}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void desassociar(@PathVariable Long formaPagamentoId, @PathVariable Long restauranteId){
        restauranteService.desassociarFormaPagamento(restauranteId, formaPagamentoId);
    }

    @PutMapping("/{formaPagamentoId}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void associar(@PathVariable Long formaPagamentoId, @PathVariable Long restauranteId){
        restauranteService.associarFormaPagamento(restauranteId, formaPagamentoId);
    }


}
```

O mapeamento ficou dessa maneira:

```
14      @RestController
15      @RequestMapping("/restaurantes/{restauranteId}/formas-pagamento")
16      public class RestauranteFormaPagamentoController {
            3 usages
17          @Autowired
```

em RestauranteService

```
95      @Transactional
96      public void desassociarFormaPagamento(Long restauranteId, Long formaPagamentoId){
97          final Restaurante restaurante = buscarOuFalhar(restauranteId);
98          final FormaPagamento formaPagamento = formaPagamentoService.buscar(formaPagamentoId);
99
100         restaurante.removerFormaPagamento(formaPagamento);
101
102     }
        1 usage
103     @Transactional
104     public void associarFormaPagamento(Long restauranteId, Long formaPagamentoId){
105         final Restaurante restaurante = buscarOuFalhar(restauranteId);
106         final FormaPagamento formaPagamento = formaPagamentoService.buscar(formaPagamentoId);
107
108         restaurante.associarFormaPagamento(formaPagamento);
109     }
110
111 }
```

e para listagem injetamos o service de FormaPagamento

12.13. Desafio- implementando os endpoints de produtos

domingo, 19 de março de 2023 21:48

- Criar DTO, input DTO para requisição POST
- Classe utilitária usando Model Mapper
- Exceção personalizada para a entidade
- Criar repositório com Query Methods e prefixos
- Classes de serviço e controladores

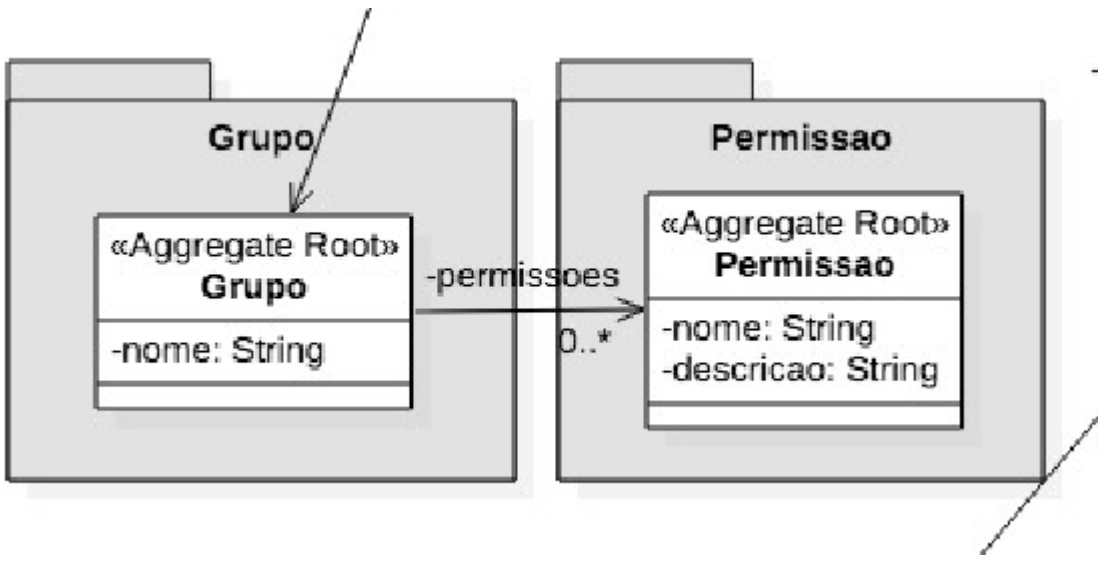
12.14. Desafio- Implementando os endpoints de abertura e fechamento de restaurantes

segunda-feira, 20 de março de 2023 20:21

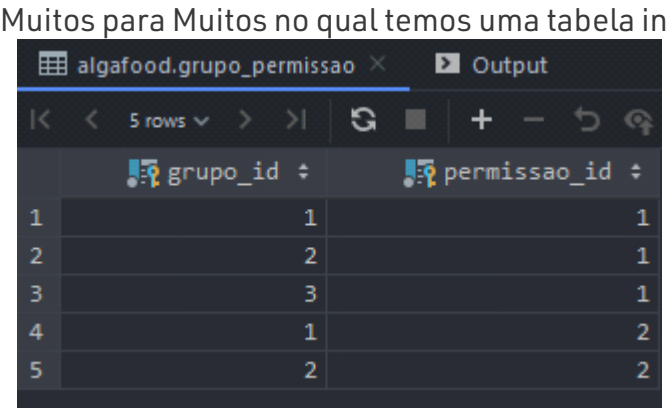
- Adicionar a coluna aberto na tabela restaurante
- Adicionar a nova propriedade na entidade
- Atualizar representação do recurso com a nova entidade
- Implementar endpoint PUT para o fechamento ou abertura
    - PUT restaurantes/restauranteId/fechamento
    - Poderíamos implementar um GET para retornar informações sobre abertura e fechamento
    - Retornar 204 NO CONTENT
- Implementar endpoint para GET na busca e listagem

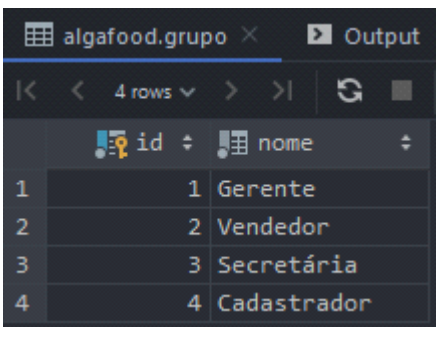
12.15. Desafio- implementando os endpoints de associação de grupos com permissões

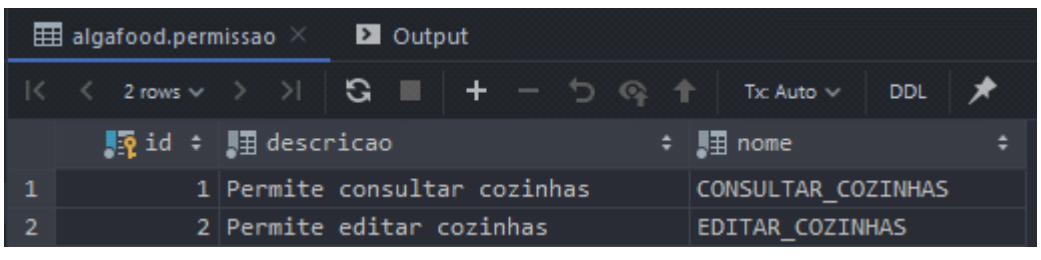
segunda-feira, 20 de março de 2023 23:31



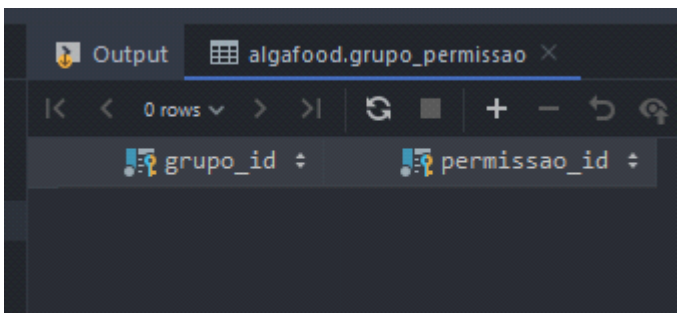
Um grupo tem muitas permissões e uma permissão faz parte de muitos grupos. Esta é uma associação de Muitos para Muitos no qual temos uma tabela intermediária.





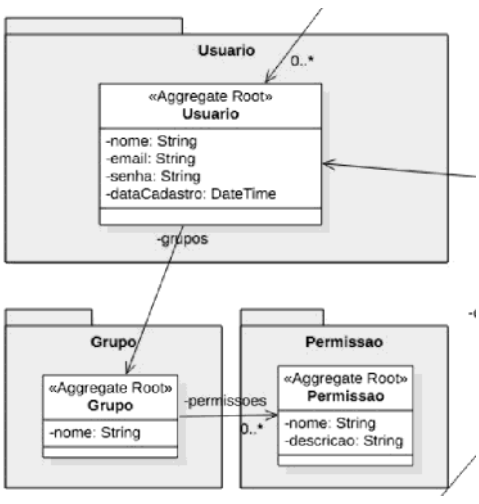


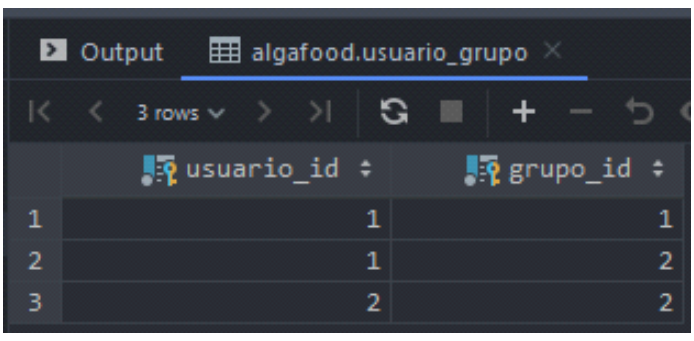
- Implementar endpoint `GET` em `grupos/grupoId/permissoes`
  - Retornar a lista de permissões que o grupo conter
- Desassociar permissões de grupos criando um endpoint `DELETE` em `grupos/grupoId/permissoes/permissoesId`
  - retornar `204 NO CONTENT`
- Associar permissões de grupos com `PUT` em `grupos/grupoId/permissoes/permissoesId`
  - retornar `204 NO CONTENT`



12.16. Desafio- implementando os endpoints de associação de usuários com grupos

terça-feira, 21 de março de 2023 14:56

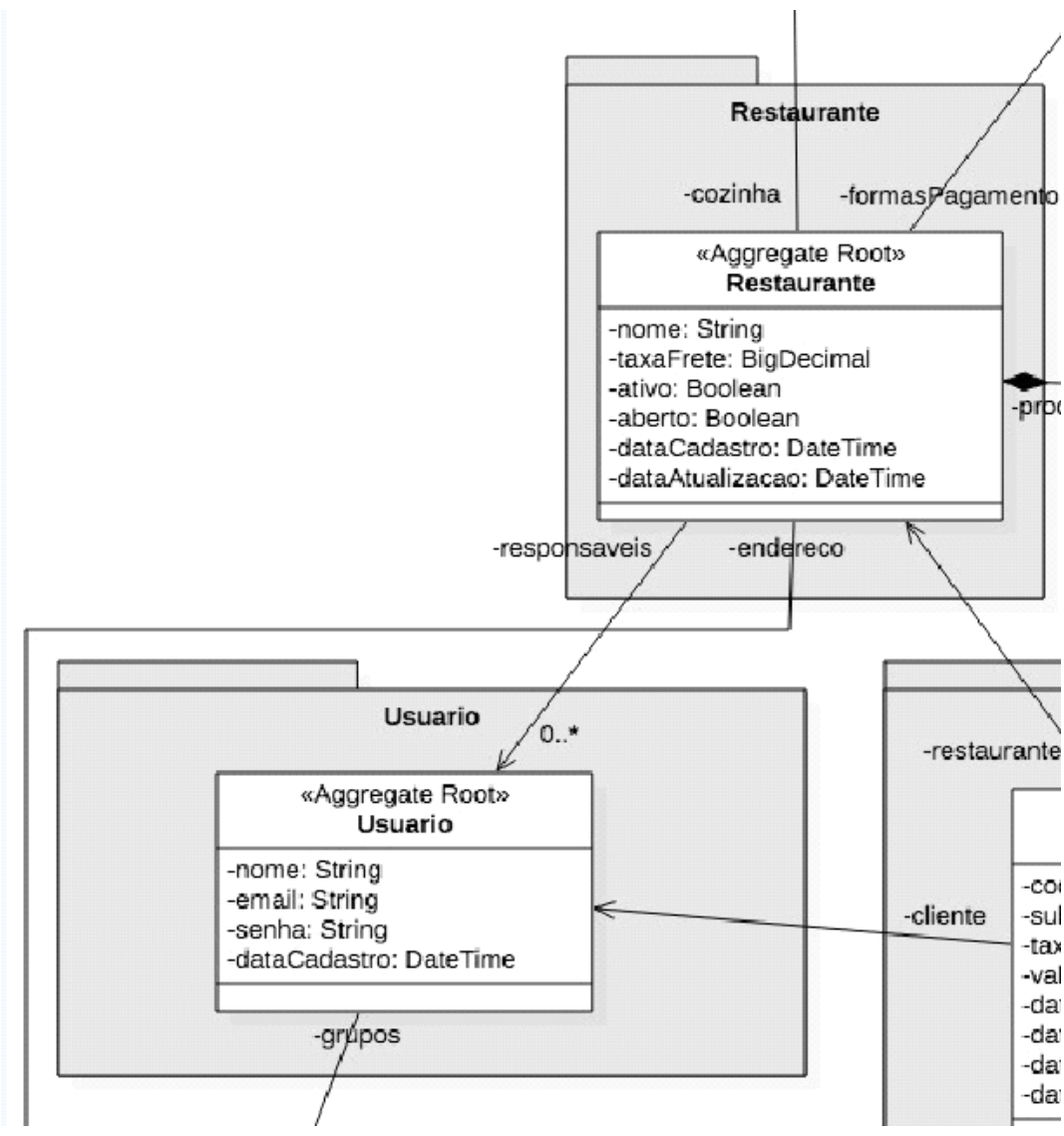




- Listar grupos de um usuário
  - GET usuários/usuárioId/grupos
- Associar um grupo a um usuário
  - PUT usuários/usuárioId/grupos/grupoId
- Desassociar um grupo de um usuário
  - DELETE usuários/usuárioId/grupos/grupoId

12.17. Desafio- implementando endpoints de associação de usuários responsáveis com restaurantes

terça-feira, 21 de março de 2023 22:40



No dominio do negócio, um restaurante pode ter vários responsáveis, e um usuário pode ser responsável por vários restaurantes. É um relacionamento Many to Many.

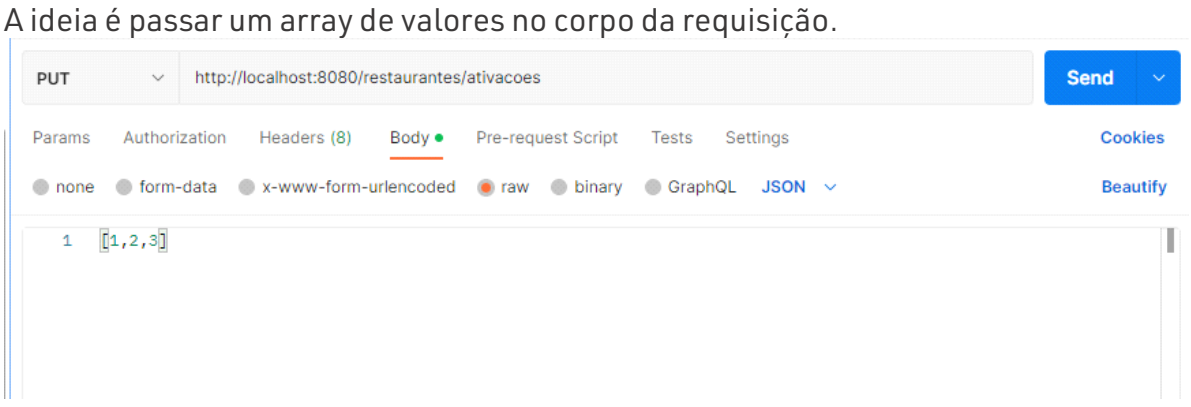
- Criar script de migração
- alterar afterMigrate
  - query para limpar o conteúdo do banco
  - query para adicionar registros na nova tabela
- Criar tabela de associação no banco de dados
  - restaurante_usuario_responsavel
- Criar carga de dados no afterMigrate
- Implementar endpoint GET para retornar uma coleção de usuários responsáveis pelo restaurante
  - GET restaurantes/restauranteId/responsaveis
  - Retornar coleção de usuários responsáveis
- Associar usuário a um restaurante
  - PUT restaurante/restauranteId/responsaveis/usuarioId
  - Retornar NO CONTENT
- Desassociar um usuário de um restaurante
  - DELETE restaurante/restauranteId/responsaveis/usuarioId

12.18. Implementando ativação e inativação em massa de restaurantes

quarta-feira, 22 de março de 2023 10:29

A operação de ativar e desativar um restaurante até então, é individual, obtendo o id do restaurante por parâmetro na URI. Agora a ideia é implementar endpoints para ativação e desativação de mais de uma entidade por requisição.

A ideia é passar um array de valores no corpo da requisição.



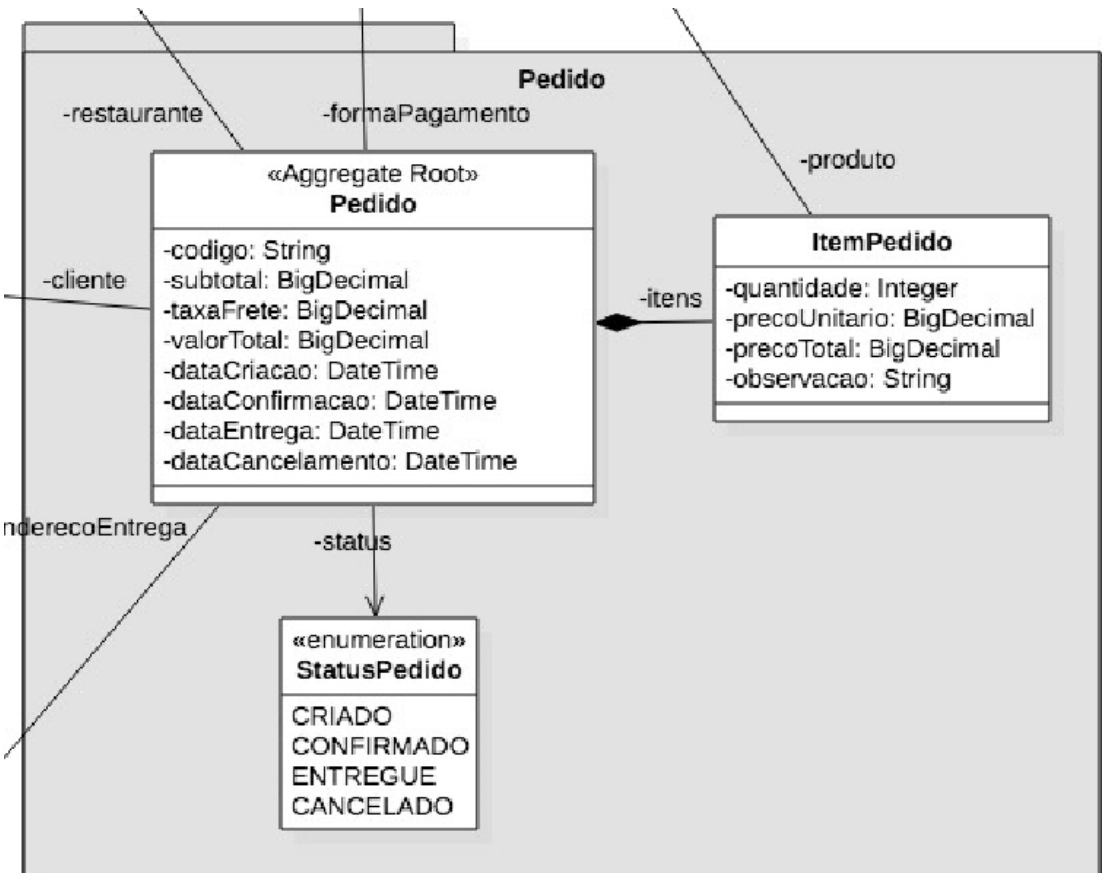
para ser recebido no método do controlador como uma coleção de valores.

```
@PutMapping("/ativacoes")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void ativarRestaurantes(@RequestBody Set<Long> restauranteIds) {
    try {
        restauranteService.ativarEmMassa(restauranteIds);
    } catch (RestauranteNaoEncontradoException e) {
        throw new NegocioException(e.getMessage(), e);
    }
}

@DeleteMapping("/inativacoes")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void inativarRestaurantes(@RequestBody Set<Long> restauranteIds) {
    try {
        restauranteService.inativarEmMassa(restauranteIds);
    } catch (RestauranteNaoEncontradoException e) {
        throw new NegocioException(e.getMessage(), e);
    }
}
```

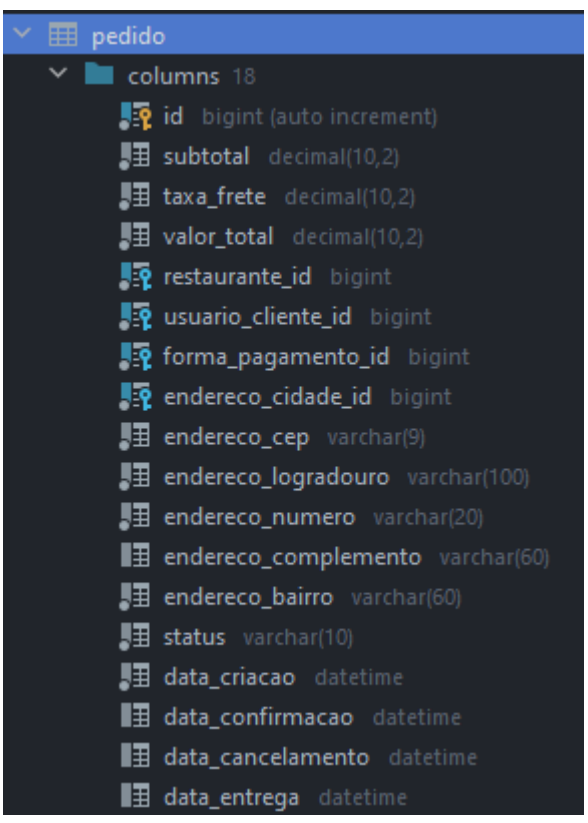
```
@Transactional
public void ativarEmMassa(Set<Long> restauranteIds){
    restauranteIds.forEach(this::ativar);
}
```

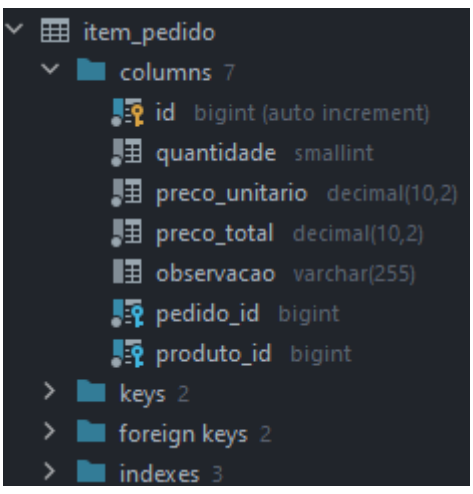
12.19. Desafio: Implementando os endpoints de consulta de pedidos

quarta-feira, 22 de março de 2023 14:12



Um pedido contém vários itens de pedido, e um item de pedido possui um pedido





- implementar endpoint para listar todos os pedidos junto com seus itens de pedido
- Adicionar querys para limpar registros da tabela pedido e item_pedido
- adicionar registros de pedido e item_pedido para testes
- Adicionar método na entidade Restaurante para verificar se um Restaurante aceita ou não uma forma de pagamento
- Adicionar método na entidade Pedido para calcular o valor total do pedido

```
public void calcularValorTotal() {
    this.subtotal = getItens().stream()
        .map(item -> item.getPrecoTotal())
        .reduce(BigDecimal.ZERO, BigDecimal::add);

    this.valorTotal = this.subtotal.add(this.taxaFrete);
}

public void definirFrete() {
    setTaxaFrete(getRestaurante().getTaxaFrete());
}

public void atribuirPedidoAosItens() {
    getItens().forEach(item -> item.setPedido(this));
}
```

- Na propriedade Status da entidade Pedido:
    - Adicionar a anotação @Enumerated do tipo STRING
- Implementar DTO para representar pedidos
    - Entidade resumida de Restaurante
    - UsuarioDTO
    - FormaPagamentoDTO
    - EnderecoDTO
    - Lista de itemPedidoDTO
- Implementar DTO para representar Itens de Pedido
- Implementar classes de conversão usando Model Mapper

12.20. Otimizando a query de pedidos e retornando model resumido na listagem

quarta-feira, 22 de março de 2023 21:32

- Resolvendo problema N+1
  - Relacionamentos ToMany são por padrão Lazy - preguiçoso e não fazem a busca antecipada no banco de dados
  - Relacionamentos ToOne por padrão são EAGER - ansioso

→ Problema: na listagem de pedidos o JPA faz consultas a mais e podemos otimizar em um só select todos os dados necessários:

```
Hibernate: select pedido0_.id as id1_7_, pedido0_.usuario_cli
Hibernate: select usuario0_.id as id1_13_0_, usuario0_.data_c
Hibernate: select formapagam0_.id as id1_3_0_, formapagam0_.d
Hibernate: select restaurant0_.id as id1_10_0_, restaurant0_.
Hibernate: select formapagam0_.id as id1_3_0_, formapagam0_.d
Hibernate: select restaurant0_.id as id1_10_0_, restaurant0_.
```

- Precisamos do select de Pedido, Usuario e Restaurante, porém, podemos otimizar os selects gerados para essas entidades.

- Não precisamos do select de formaPagamento, além da outra consulta de Restaurante.
- Solução: Join Fetch para quando buscar um Pedido, realizar Join na tabela Usuario e restaurante

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

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

O JPA não faz selects nas tabelas separadamente, poupando processamento no banco de dados

```
Hibernate: select pedido0_.id as id1_7_
Hibernate: select formapagam0_.id as id
Hibernate: select cozinha0_.id as id1_1_
Hibernate: select formapagam0_.id as id
Hibernate: select cozinha0_.id as id1_1_
```

Em contra partida, FormaPagamento e Cozinha estão sendo consultadas no banco de dados, pois esses relacionamentos são ToOne padronizados com EAGER, podemos anotar esses relacionamentos com a propriedade fetch e alterando a estratégia de fetching para Lazy.

```
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(nullable = false)
private FormaPagamento formaPagamento;
```

O JPA está buscando também cozinha, pois dentro de Restaurante(ToOne) contém Cozinha(ToOne) porém, é importante decidir com lógica a alteração do fetching para não afetar outros pontos da aplicação. Não iremos alterar pois está distante do domínio de Pedidos e podemos nos precipitar, a alternativa temporária é incluir a busca de Cozinha na Query do Repositório de Pedidos.

```
@Repository
public interface PedidoRepository extends Custo

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

Agora o JPA faz somente uma consulta ao banco

```
2023-03-23 01:20:11.709  INFO 3744 --- [nio
Hibernate: select pedido0_.id as id1_7_0_,
```

- Limpar as informações na resposta de uma lista de Pedidos
  - Deixar intacto a busca de pedido, pois é coerente ter informações adicionais sobre um pedido individual

- Adicionar um DTO específico para listagem de pedidos com menos informações
- Adicionar classe que converte entidades e DTOs com Model Mapper

12.21. Desafio- Implementando o endpoint de emissão de pedidos

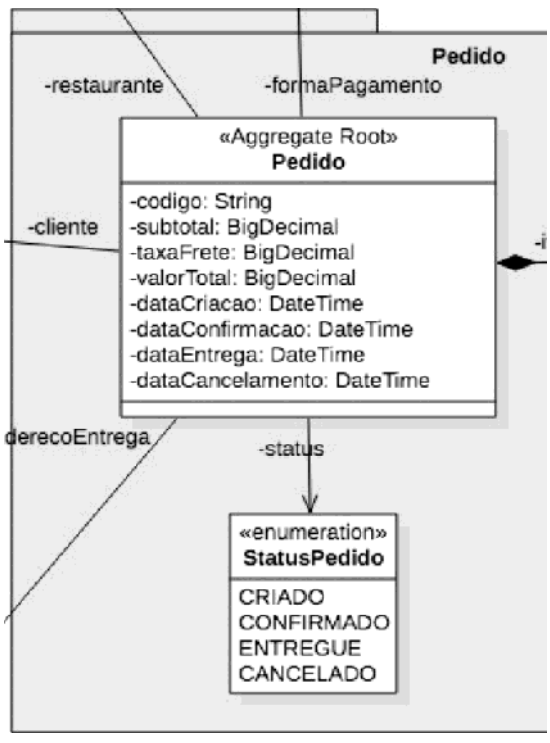
quarta-feira, 22 de março de 2023 22:26

- Implementar endpoint POST
    - /pedidos. Payload:
        - restaurante: id
        - formaPagamento: id
        - enderecoEntrega: enderecoInput
        - ItemPedidoInput
            - ProdutoId
            - Quantidade
            - Observação

- Adicionar em cascata itens de pedido em Pedido
- Ignorar mapeamento do modelmapper configurando manualmente na definição do Bean
    - ItemPedidoInput para ItemPedido
        - addMapping passando instancia do mapper em uma expressão lambda e chamado o método skip.

12.22. Implementando endpoint de transição de status de pedidos

sexta-feira, 24 de março de 2023 14:27



- Implementar regra de Status do Pedido e fazer transição de estados no Status.
  - Quando um pedido é criado, é atribuído automaticamente o status CRIADO.
  - Quando um restaurante aceita o Pedido, ele é alterado para CONFIRMADO.
  - Quando o Pedido é entregue, o status é alterado para ENTREGUE.
  - Um status cancelado é atribuído quando um Pedido não foi CONFIRMADO.

Pensando no processo de negócio:

A ação de alterar pedido é um processo de negócio, podemos criar endpoints desse processo. Exemplos de implementação do processo de negócio:

- Sem payload na requisição:
  - PUT /pedidos/pedidoId/confirmacao
  - PUT /pedidos/pedidoId/entrega
  - PUT /pedidos/pedidoId/cancelado

- Com payload na requisição
  - PUT /pedidos/pedidoId/status
    - Corpo: CANCELADO, CONFIRMADO ou ENTREGUE
    - Recebido no controlador: constante String

- Submetendo uma requisição POST com um status:
  - POST pedidos/1/alteracoes-status
    - Corpo: JSON {"status":"CONFIRMADO"}
  - GET pedidos/1/alteracoes-pedidos
    - [

```
{
    "status":"CRIADO",
    "dataAlteracao":"2023-11-08T22:00:55Z"
},
{
    "status":"CONFIRMADO",
    "dataAlteracao":"2023-11-08T22:02:00Z"
}
,
{
    "status":"ENTREGUE",
    "dataAlteracao":"2023-11-08T22:22:00Z"
}
]
```

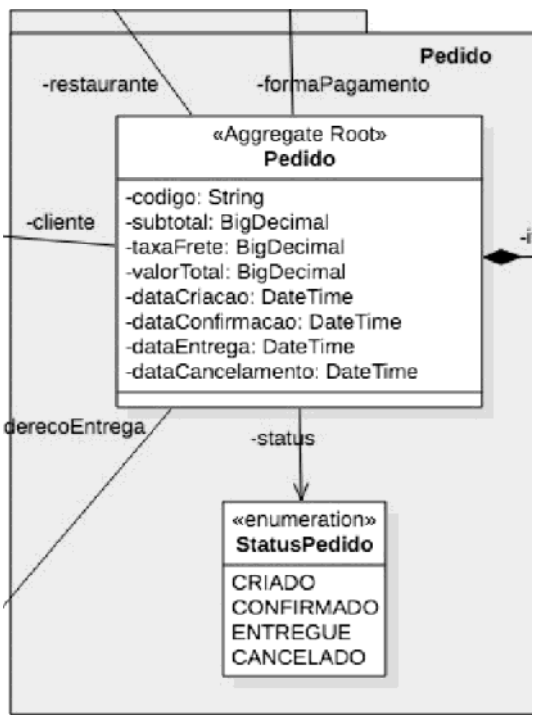
Implementação:

```java
@Transactional
public void confirmar(Long pedidoId){
    final Pedido pedido = pedidoService.buscar(pedidoId);
    if(!pedido.getStatus().equals(StatusPedido.CRIADO)){
        throw new NegocioException(String.format(
                ErrorMessage.STATUS_PEDIDO.get(),
                pedido.getId(),
                pedido.getStatus().getStatus(),
                StatusPedido.CONFIRMADO.getStatus()));
    }
    pedido.setStatus(StatusPedido.CONFIRMADO);
    pedido.setDataConfirmacao(OffsetDateTime.now());
}
```

```java
@PutMapping("/confirmacao")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void confirmar(@PathVariable Long pedidoId){
    fluxoPedidoService.confirmar(pedidoId);
}
```

12.23. Desafio- implementando endpoints de transição de status de pedidos

sexta-feira, 24 de março de 2023 23:31



- Implementar regra de Status do Pedido e fazer transição de estados no Status.
  - Quando um pedido é criado, é atribuído automaticamente o status CRIADO.
  - Quando um restaurante aceita o Pedido, ele é alterado para CONFIRMADO.
  - Quando o Pedido é entregue, o status é alterado para ENTREGUE.
  - Um status cancelado é atribuído quando um Pedido não foi CONFIRMADO.

Desafio: Implementar endpoints de Entregue e Cancelado

```java
@PutMapping("/entrega")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void entregar(@PathVariable Long pedidoId) { fluxoPedidoService.entregar(pedidoId); }

@PutMapping("/cancelamento")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void cancelar(@PathVariable Long pedidoId) { fluxoPedidoService.cancelar(pedidoId); }
```

```java
    @Transactional
    public void confirmar(Long pedidoId){
        final Pedido pedido = pedidoService.buscar(pedidoId);
        if(!pedido.getStatus().equals(StatusPedido.CRIADO)){
            throw new NegocioException(String.format(
                    ErrorMessage.STATUS_PEDIDO.get(),
                    pedido.getId(),
                    pedido.getStatus().getStatus(),
                    StatusPedido.CONFIRMADO.getStatus()));
        }
        pedido.setStatus(StatusPedido.CONFIRMADO);
        pedido.setDataConfirmacao(OffsetDateTime.now());
    }

    1 usage
    @Transactional
    public void entregar(Long pedidoId){
        final Pedido pedido = pedidoService.buscar(pedidoId);
        if(!pedido.getStatus().equals(StatusPedido.CONFIRMADO)){
            throw new NegocioException(String.format(
                    ErrorMessage.STATUS_PEDIDO.get(),
                    pedido.getId(),
                    pedido.getStatus().getStatus(),
                    StatusPedido.ENTREGUE.getStatus()));
        }
        pedido.setStatus(StatusPedido.ENTREGUE);
        pedido.setDataEntrega(OffsetDateTime.now());
    }

    1 usage
    @Transactional
    public void cancelar(Long pedidoId) {
        final Pedido pedido = pedidoService.buscar(pedidoId);
        if(!pedido.getStatus().equals(StatusPedido.CRIADO)){
            throw new NegocioException(String.format(
                    ErrorMessage.STATUS_PEDIDO.get(),
                    pedido.getId(),
                    pedido.getStatus().getStatus(),
                    StatusPedido.CANCELADO.getStatus()));
        }
        pedido.setStatus(StatusPedido.CANCELADO);
        pedido.setDataCancelamento(OffsetDateTime.now());
    }
}
```

12.24. Refatorando o código de regras para transição de status de pedidos

sábado, 25 de março de 2023 00:20

Na aula 12.23. Desafio- implementando endpoints de transição de status de pedidos fizemos métodos do processo de negócio para alterar status de um pedido. Mas ainda assim, podemos melhorar a legibilidade do código e delegar tarefas da classe de serviço para a classe da entidade, tornando-a uma entidade rica. Uma entidade rica é uma entidade que possui métodos utilitários com processos de negócio que possuem responsabilidades específicas para a entidade, como calcularPrecoTotal na classe Restaurante.

```
1 usage
public void calcularValorTotal(){
    this.subTotal = itensPedido.stream()
            .map(ItemPedido::getPrecoTotal)
            .reduce(BigDecimal.ZERO, BigDecimal::add);

    this.valorTotal = this.subTotal.add(this.taxaFrete);
}
```

12.24. Refatorando o código de regras para transição de status de pedidos

domingo, 26 de março de 2023 20:14

Na classe de Pedido foram implementados os métodos de alterações de pedido, tornando-a uma entidade rica:

```java
@Service
public class FluxoPedidoService {

    3 usages
    @Autowired
    private PedidoService pedidoService;

    1 usage
    @Transactional
    public void confirmar(Long pedidoId){
        final Pedido pedido = pedidoService.buscar(pedidoId);
        pedido.confirmar();
    }

    1 usage
    @Transactional
    public void entregar(Long pedidoId){
        final Pedido pedido = pedidoService.buscar(pedidoId);
        pedido.entregar();
    }

    1 usage
    @Transactional
    public void cancelar(Long pedidoId) {
        final Pedido pedido = pedidoService.buscar(pedidoId);
        pedido.cancelar();
    }
}
```

Para cada método dentro da classe, chamamos o setStatus criado manualmente (sobrescrevendo a implementação do Lombok)

```
1 usage
public void confirmar(){
    setStatus(StatusPedido.CONFIRMADO);
    setDataConfirmacao(OffsetDateTime.now());
}

1 usage
public void cancelar(){
    setStatus(StatusPedido.CANCELADO);
    setDataCancelamento(OffsetDateTime.now());
}
1 usage
public void entregar(){
    setStatus(StatusPedido.ENTREGUE);
    setDataEntrega(OffsetDateTime.now());
}
```

O método setStatus ficou responsável de lançar a exception de acordo com tipo de status que foi tentado alterar.

```
3 usages
private void setStatus(StatusPedido novoStatus){
    if(getStatus().naoPodeAlterarPara(novoStatus)){
        throw new NegocioException(String.format(
                ErrorMessage.STATUS_PEDIDO.get(),
                getId(),
                this.getStatus().getStatus(),
                novoStatus.getStatus()));
    }
```

a lógica do processo de alteração de status foi delegada para dentro da Enumeração Status. Basicamente: se o Status atual não puder ser alterado para o NovoStatus, lança a exceção com a mensagem, Status do pedido "x" não pode ser alterado de "Criado" para "Entregue".

```java
12 usages
@Getter
public enum StatusPedido {

    3 usages
    CRIADO( valor: 1,  status: "Criado"),
    2 usages
    CONFIRMADO( valor: 2,  status: "Confirmado", CRIADO),
    1 usage
    ENTREGUE( valor: 3,  status: "Entregue", CONFIRMADO),
    1 usage
    CANCELADO( valor: 4,  status: "Cancelado", CRIADO);

    1 usage
    private final String status;
    1 usage
    private final int valor;

    2 usages
    private List<StatusPedido> statusAnteriores;

    4 usages
    private StatusPedido(int valor, String status, StatusPedido... statusAnteriores){
        this.valor = valor;
        this.status = status;
        this.statusAnteriores = Arrays.asList(statusAnteriores);
    }

    1 usage
    public boolean naoPodeAlterarPara(StatusPedido novoStatus){
        return !novoStatus.statusAnteriores.contains(this);
    }
}
```

dentro do Status atual contém uma lista que contém o Status anterior que é necessário para chegar até o Status atual. Se dentro do novoStatus não conter na lista dele o Status atual, retornará verdadeiro e entrará no if lançando a exception.