

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli



The top of the slide features a decorative header with a dark red background. It contains several overlapping geometric patterns: concentric circles on the left, a series of parallel diagonal lines in the center, and a semi-circle on the right.

ENTIDADES

Mapeamento e Relacionamentos

Aula 08

ENTIDADES E O BANCO DE DADOS



- Várias entidades ainda precisam ser criadas do projeto Help Desk;
- Além disso, os relacionamentos também devem ser considerados na criação das classes / tabelas;
- A vantagem (ou não - a depender do projeto) do uso do framework é que não precisamos manipular o banco de dados diretamente;
- A criação do correto mapeamento das entidades, na primeira execução, será capaz de criar as tabelas diretamente na base de dados;
- Assim, preocupe-se apenas com a codificação (correta) das entidades.

TIPO STATUS

```
3 public enum TipoStatus { 4 usages
4
5     AGUARDANDO_ATENDIMENTO( descricao: "Aguardando Atendimento"), 1 usage
6     INICIADO( descricao: "Em atendimento"), no usages
7     ENCAMINHADO( descricao: "Encaminhado"), no usages
8     DEVOLVIDO( descricao: "Devolvido"), no usages
9     CONCLUIDO( descricao: "Concluído"); no usages
10
11     private final String descricao; 2 usages
12
13     > private TipoStatus(String descricao) { this.descricao = descricao; }
14
15
16
17     @Override
18     > public String toString() { return descricao; }
19
20
21
22 }
```

- Garante que os valores sejam padronizados;
- Sempre utilizaremos esses valores para a coluna de status da respectiva classe.

STATUS

```
10  @Data 7 usages
11  @Entity
12  @Table(name = "status", schema = "public")
13   public class Status implements Serializable {
14
15      @Serial
16      private static final long serialVersionUID = 4754738448382743837L;
17
18      @Id
19      @Column(name = "status", nullable = false, unique = true)
20      @Enumerated(EnumType.STRING)
21       private TipoStatus status;
22
23  }
```

- O id é o próprio valor do status;
- Não importa um id nesse caso, mas sim, o status padronizado e via banco

ARQUIVO

```
10  @Data 4 usages
11  @Entity
12  @Table(name = "arquivo", schema = "public")
13  public class Arquivo implements Serializable {
14
15      @Serial
16      public static final long serialVersionUID = -2732822372828732899L;
17
18      @Id
19      @SequenceGenerator(
20          name = "SEQ-ARQUIVO",
21          sequenceName = "public.seq_arquivo",
22          allocationSize = 1)
23      @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ-ARQUIVO")
24      private Long id;
25
26      @Column(name = "nome")
27      private String nome;
28
29      @Column(name = "caminho")
30      private String caminho;
```

ARQUIVO

```
32     @Column(name = "mime_type")
33     (a) private String mimeType;
34
35     @Column(name = "tamanho")
36     (a) private Integer tamanho;
37
38     @Transient
39     private Resource conteudoResource;
40
41 }
```

- @Transient: um atributo que é necessário para a codificação da classe, mas que não deve ser persistido no banco de dados.

ATENDIMENTO

```
10  @Data 3 usages
11  @Entity
12  @Table(name = "atendimento", schema = "public")
13  public class Atendimento implements Serializable {
14
15      @Serial
16      private static final long serialVersionUID = 9564223487458874L;
17
18      @Id
19      @SequenceGenerator(
20          name = "SEQ-ATENDIMENTO",
21          sequenceName = "public.seq_atendimento",
22          allocationSize = 1)
23      @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ-ATENDIMENTO")
24      private Long id;
25
26      @ManyToOne
27      @JoinColumn(name = "status", referencedColumnName = "status", nullable = false)
28      private Status status;
29
30      @ManyToOne
31      @JoinColumn(name = "id_usuario_equipe", columnDefinition = "id", nullable = false)
32      private UsuarioEquipe usuarioEquipe;
```


ATENDIMENTO

```
34     @ManyToOne
35     @JoinColumn(name = "id_ticket", referencedColumnName = "id", nullable = false)
36     private Ticket ticket;
37
38     @Column(name = "data_atendimento", nullable = false)
39     @Temporal(TemporalType.TIMESTAMP)
40     private LocalDateTime dataAtendimento;
41
42     @Column(name = "descricao_atendimento", nullable = false, columnDefinition = "TEXT", length = 500)
43     private String descricaoAtendimento;
44
45     @ManyToOne
46     @JoinColumn(name = "id_arquivo", referencedColumnName = "id")
47     private Arquivo arquivo;
48
49 }
```

TICKET

```
12 @Data 3 usages
13 @Entity
14 @Table(name="ticket", schema = "public")
15 public class Ticket implements Serializable {
16
17     @Serial
18     private static final long serialVersionUID = -928347548287458874L;
19
20     @Id
21     @SequenceGenerator(
22         name = "SEQ-TICKET",
23         sequenceName = "public.seq_ticket",
24         allocationSize = 1)
25     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ-TICKET")
26     private Long id;
27
28     @ManyToOne
29     @JoinColumn(name = "id_usuario", referencedColumnName = "id", nullable = false)
30     private Usuario solicitante;
31
32     @Column(name = "data_abertura", nullable = false)
33     @Temporal(TemporalType.TIMESTAMP)
34     private LocalDateTime dataAbertura;
```

TICKET

```
36     @Column(name = "titulo", length = 50, nullable = false)
37     private String titulo;
38
39     @Column(name = "descricao", length = 1000, columnDefinition = "TEXT", nullable = false)
40     private String descricao;
41
42     @ManyToOne
43     @JoinColumn(name = "status", referencedColumnName = "status")
44     private Status ultimoStatus;
45
46     @ManyToOne
47     @JoinColumn(name = "id_arquivo", referencedColumnName = "id")
48     private Arquivo arquivo;
49
50     @Column(name = "relatorio_interno", length = 1000, columnDefinition = "TEXT")
51     private String relatorioInterno;
```

TICKET

```
53     @Column(name = "data_conclusao")
54     @Temporal(TemporalType.TIMESTAMP)
55     private LocalDateTime dataConclusao;
56
57     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
58     @JoinColumn(name = "id_ticket")
59     private List<Atendimento> atendimentos;
60
61     @ManyToOne
62     @JoinColumn(name = "id_equipe", referencedColumnName = "id")
63     private Equipe equipe;
64
65 }
```

- nesse modelo, iremos desprezar a classe de Categoria.


- `@Temporal (TemporalType.TIMESTAMP)` : indica indica que um atributo Java deve ser mapeado como um valor de data e hora completo (data, hora, segundos, etc.) no banco de dados.






CRIANDO REPOSITORIES

- Gere todas as classes de repositórios das novas entidades
- Os serviços, controladores, dtos serão criados especificamente em cada contexto
- Atenção com StatusRepository

```
public interface StatusRepository extends JpaRepository<Status, TipoStatus> {  
}
```



▼  repositories

-  ArquivoRepository
-  AtendimentoRepository
-  CategoriaRepository
-  EquipeRepository
-  PerfilRepository
-  StatusRepository
-  TicketRepository
-  UsuarioRepository

EXECUTANDO

- Execute o projeto
- Verifique no Banco de Dados a criação das tabelas
- Verifique o relacionamento entre as tabelas
- Confira a criação das sequences

The top of the image features a dark red header with a series of repeating geometric patterns. These patterns include concentric circles, semi-circles, and radial lines, all rendered in a slightly lighter shade of red than the background.

CONTEXTO

Abrir novo Ticket

DTO

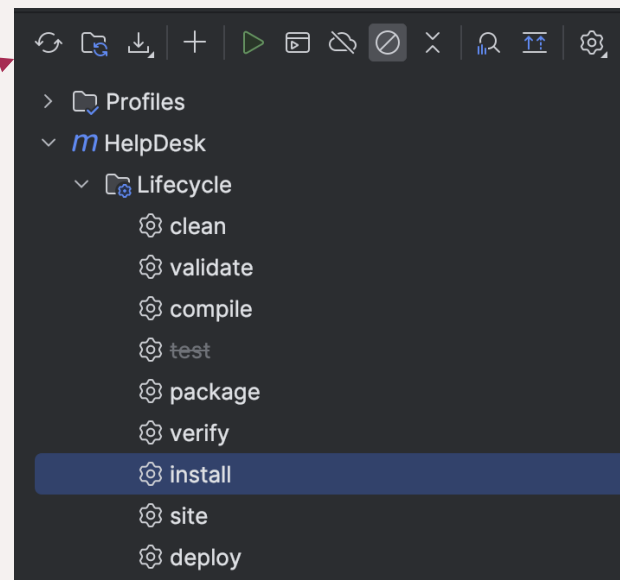
- Recebe os dados básicos com a descrição do problema;
- Note que esse DTO não possui nenhum método;
 - É preciso buscar usuário que está abrindo o chamado e a equipe que o chamado está sendo direcionado;
 - DTO não executa buscas na base

```
8      public record AbrirTicketDTO( no usages
9
10         Long id_usuario, no usages
11         String titulo, no usages
12         String descricao, no usages
13         Long id_equipe no usages
14
15     ) implements Serializable {
16
17         @Serial
18         public static final long serialVersionUID = 4326482468263482L;
19
20     }
```


DTO

- A validação pode ser feita diretamente no DTO
- Isso impede excesso de código ... != null
- Adicione a biblioteca Bean Validation no pom.xml
- No Maven, clique em Install
- No Maven, clique em atualizar

```
64      <dependency>  
65          <groupId>javax.validation</groupId>  
66          <artifactId>validation-api</artifactId>  
67          <version>2.0.1.Final</version>  
68      </dependency>
```



DTO

- Adicione a anotação `@NotEmpty`: garante que o valor não poder ser NULO e nem estar vazio;
- `@NotNull` : valida apenas para a situação de NULL

```
9      public record AbrirTicketDTO( 2 usages
10
11          @NotEmpty(message = "Usuário não pode ser nulo.") 1 usage
12          Long id_usuario,
13
14          @NotEmpty(message = "Título é um campo obrigatório.") 1 usag
15          String titulo,
16
17          @NotEmpty(message = "Descrição é um campo obrigatório.") 1
18          String descricao,
19
20          @NotEmpty(message = "Equipe é obrigatória ser informada.")
21          Long id_equipe
22
23      ) implements Serializable {
24
25          @Serial
26          public static final long serialVersionUID = 4326482468263482L;
27
28      }
```

SERVICE

- Deve buscar o usuário solicitante
- Deve buscar a equipe de direcionamento do ticket
- Ainda cria um novo status
- Adiciona a Data/Hora atual
- Salva o objeto criado

```
16 @Service 1 usage
17 public class AbrirTicketService {
18
19     @Autowired
20     TicketRepository ticketRepository;
21
22     @Autowired
23     UsuarioRepository usuarioRepository;
24
25     @Autowired
26     EquipeRepository equipeRepository;
27
28     @
29     public Ticket abrirTicket(AbrirTicketDTO dto) { 1 usage
30
31         Usuario solicitante = usuarioRepository.findById(dto.id_usuario()).orElse( other: null);
32         Equipe equipe = equipeRepository.findById(dto.id_equipe()).orElse( other: null);
```

SERVICE

```
33 @Transactional 1 usage
34 @
35 public Ticket abrirTicket(AbrirTicketDTO dto) {
36     Usuario solicitante = usuarioRepository.findById(dto.id_usuario()).orElse( other: null);
37     Equipe equipe = equipeRepository.findById(dto.id_equipe()).orElse( other: null);
38
39     if (solicitante != null && equipe != null) {
40         Ticket ticket = new Ticket();
41         ticket.setSolicitante(solicitante);
42         ticket.setTitulo(dto.titulo());
43         ticket.setDescricao(dto.descricao());
44         ticket.setDataAbertura(LocalDate.now());
45         ticket.setEquipe(equipe);
46
47         Status status = new Status();
48         status.setStatus(TipoStatus.AGUARDANDO_ATENDIMENTO);
49         ticket.setUltimoStatus(status);
50
51         return ticketRepository.save(ticket);
52     }
53
54     return null;
55 }
56 }
```

- @Transactional
- Indica que várias operações de save / update deverão ser feitas;
- Se uma falhar, automaticamente é efetuado rollback;
- Se tudo der certo, um commit é executado;
- Necessário utilizar para salvar tanto o Ticket quanto o valor de Status (quando não estiver gravado).

CONTROLLER

- Devolve para o cliente apenas o status da requisição;
- O próprio DTO faz a validação no ato da requisição;
- Object representa "qualquer coisa"

```
8      @RestController
9      @RequestMapping("/abrirTicket")
10     public class AbrirTicketController {
11
12         @Autowired
13         AbrirTicketService abrirTicketService;
14
15         @PostMapping("/add")
16         public ResponseEntity<Object> abrirTicket(@RequestBody AbrirTicketDTO dto) {
17             if (dto != null) {
18                 if (abrirTicketService.abrirTicket(dto) != null) {
19                     return ResponseEntity.status(HttpStatus.CREATED).build();
20                 }
21             }
22             return ResponseEntity.status(HttpStatus.UNPROCESSABLE_ENTITY).build();
23         }
24
25     }
```

Teste

The screenshot displays a REST client interface with a dark theme. At the top, a dropdown menu shows the HTTP method 'POST' and the URL 'http://localhost:8080/HelpDesk/api/abrirTicket/add'. To the right is a blue 'Send' button. Below the URL bar, a series of tabs are visible: 'Params', 'Authorization', 'Headers (8)', 'Body' (which is selected and underlined), 'Scripts', and 'Settings'. On the far right of this section are 'Cookies' and 'Beautify' links. Under the 'Body' tab, there are radio buttons for different content types: 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected), 'binary', and 'GraphQL'. To the right of these is a 'JSON' dropdown menu. The main area of the interface contains a JSON body with the following structure:

```
1 {  
2   "id_usuario": 5,  
3   "titulo": "Primeiro chamado",  
4   "descricao" : "teste de envio",  
5   "id_equipe" : 1  
6 }
```

At the bottom of the interface, there is a status bar. It includes tabs for 'Body' (selected), 'Cookies', 'Headers (4)', and 'Test Results'. To the right of these tabs is a refresh icon. Further right, a green box displays the status '201 Created', followed by response details: '187 ms', '128 B', a globe icon, and a three-dot menu icon.

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli

