

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli



The top of the slide features a dark red header with a series of overlapping semi-circles. Each semi-circle contains a different pattern: concentric lines, a grid of dots, or a series of parallel lines.

USUÁRIO

CRUD

Aula 05

VISÃO GERAL

- A entidade Usuário possui particularidades em relação a Perfil e Equipe;
- Um usuário está obrigatoriamente associado a um perfil;
- Um usuário pode estar associado a uma equipe;
- OBJETIVO: mapear o relacionamento via programação (com @anotações) e persistir os dados de forma correta.

ENTIDADE USUÁRIO

```
14 @Table(name="usuario", schema = "public")
15 public class Usuario implements Serializable {
16
17     @Serial
18     public static final long serialVersionUID = -421308274897239279L;
19
20     @Id
21     @SequenceGenerator(
22         name = "SEQ-USUARIO",
23         sequenceName = "public.seq_usuario",
24         allocationSize = 1)
25     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ-USUARIO")
26     private Long id;
27
28     @Column(name = "nome", nullable = false, length = 255)
29     private String nome;
30
31     @Column(name = "email", nullable = false, length = 100)
32     private String email;
33
34     @Column(name = "senha", nullable = false, length = 255)
35     private String senha;
36
37     @Column(name = "cargo", nullable = false, length = 80)
38     private String cargo;
```

```
40     @Column(name = "telefone", nullable = false, length = 30)
41     private String telefone;
42
43     @OneToOne
44     @JoinColumn(name = "id_perfil", referencedColumnName = "id",
45         insertable = true, updatable = true)
46     private Perfil perfil;
47
48     @OneToMany
49     @JoinTable(name = "usuario_equipe", schema = "public",
50         joinColumns = @JoinColumn(name = "id_usuario"),
51         inverseJoinColumns = @JoinColumn(name = "id_equipe"))
52     @JsonManagedReference
53     private Collection<Equipe> equipes;
54
55
56 }
```

ENTIDADE USUÁRIO


- O QUE MUDA?

```
43     @OneToOne
44     @JoinColumn(name = "id_perfil", referencedColumnName = "id",
45                 insertable = true, updatable = true)
46     private Perfil perfil;
47
```

- Ao invés de possuir um Long para o id do perfil, fazemos referência diretamente para a classe de perfil.
- @OneToOne : um usuário só poderá ter um único perfil
- Insertable e updatable: sinaliza que, ao salvar um usuário, a relação com perfil também precisa ser criada / atualizada (a coluna `id_perfil` precisa ser preenchida com valor).

REPOSITÓRIO

- Apenas o repositório da entidade Usuário será utilizado.

```
8  
9  public interface UsuarioRepository extends JpaRepository<Usuario, Long> { 2 usages  
10  
11  
12  
13 }  
14
```

SERVIÇO

- Apenas o Serviço da entidade Usuario será criado

```
11  @Service 5 usages
12  public class UsuarioService {
13
14      @Autowired
15      UsuarioRepository usuarioRepository;
16
17
18
19
20  }
```

CONTROLLER

- Será responsável por gerenciar o CRUD da entidade Usuario
 - CRUD: Create / Read / Update / Delete

```
13  @RestController
14  @RequestMapping("/usuario")
15  public class UsuarioController {
16
17      @Autowired
18      UsuarioService usuarioService;
19
20
21
22  }
```


DTO USUÁRIO

- Nesse momento, resgata todos os campos do usuário, sem distinção.
- Note que o usuário possui o Perfil associado. Nesse caso, aplica-se o REUSO do PerfilDTO.

```
9  ✓ public record UsuarioDTO( 3 usages
10
11      Long id, 1 usage
12      String nome, 1 usage
13      String email, 1 usage
14      String senha, 1 usage
15      String cargo, 1 usage
16      String telefone, 1 usage
17      PerfilDTO perfil 3 usages
18
19  ) implements Serializable {
20
21      @Serial no usages
22      public static final long serialVersionUID = 4021309213882379399L;
```

DTO USUÁRIO

```
24     public static UsuarioDTO valueOf(Usuario usuario) { no usages
25         if (usuario != null) {
26             return new UsuarioDTO(
27                 usuario.getId(),
28                 usuario.getNome(),
29                 usuario.getEmail(),
30                 usuario.getSenha(),
31                 usuario.getCargo(),
32                 usuario.getTelefone(),
33                 PerfilDTO.valueOf(usuario.getPerfil())
34             );
35         }
36         return null;
37     }
```

DTO USUÁRIO

```
39      public static Usuario toUsuario(UsuarioDTO dto) { 2 usages
40          if (dto != null) {
41              Usuario usuario = new Usuario();
42              usuario.setId(dto.id);
43              usuario.setNome(dto.nome);
44              usuario.setEmail(dto.email);
45              usuario.setSenha(dto.senha);
46              usuario.setCargo(dto.cargo);
47              usuario.setTelefone(dto.telefone);
48              if (dto.perfil != null) {
49                  usuario.setPerfil(PerfilDTO.toPerfil(dto.perfil));
50              }
51              else usuario.setPerfil(null);
52
53              return usuario;
54          }
55          return null;
56      }
```

The top of the slide features a dark red header with a series of overlapping semi-circles. Each semi-circle contains a different pattern: concentric lines, a grid of dots, or radiating lines.

C – Create / INSERT

Cadastrar Usuários

CONTROLLER

```
20 @PostMapping("/add")
21 public @ResponseBody ResponseEntity<UsuarioDTO> cadastrar(@RequestBody UsuarioDTO dto) {
22     Usuario usuario = usuarioService.cadastrar(UsuarioDTO.toUsuario(dto));
23     return ResponseEntity.ok().body(UsuarioDTO.valueOf(usuario));
24 }
```

- Método para adicionar um novo usuário
- Os parâmetros não são passados pela URL, mas sim no corpo da página
 - Um arquivo JSON é utilizado para injetar os dados na API, via controller

SERVICE

```
17     public Usuario cadastrar(Usuario usuario) { 2 usages
18         if (usuario != null) {
19             return usuarioRepository.save(usuario);
20         }
21         return null;
22     }
```

- Nenhum recurso inovador é utilizado, quando comparado com os demais serviços já construídos

JSON

- JSON (*JavaScript Object Notation*) é um formato de dados que permite armazenar e trocar informações de forma legível por humanos e analisável por máquinas;
- É um formato aberto e independente de linguagem de programação;
- Armazenar dados temporários, como dados gerados pelo usuário;
- Transferir dados entre um servidor e um aplicativo web;
- Enviar e receber dados entre diferentes sistemas (é uma alternativa simples e mais leve ao XML);
- Exemplos: <https://www.oracle.com/br/database/what-is-json/>

```
{
  "user": {
    "id": "11",
    "animal": "Bob",
    "idade": "2"
  }
}
```

NA PRÁTICA

- Utilizando o Postman e uma pesquisa sobre o formato JSON, alimente as tabelas com os seguintes dados:
- Usuário
 - Karina Almeida, karina.almeida@fatec.br, karina1234, gerente de projetos, (11) 88785-2394, Gerente
 - Adriano Silva, adriano.silva@fatec.br, adriano1234, suporte ao usuário, (11) 92728-3843, Atendente
 - Gilberto Soares, gil.soares@fatec.br, gil1234, analista, (11) 5426-2963, Atendente
 - Michaela Santos, mik.santos@fatec.br, mik1234, analista, (11) 5426-2842, Atendente
 - Bruno Henrique Tavares, bru.tavares@fatec.br, bru1234, coordenador, (11) 5426-9098, Solicitante

TESTE


HTTP HelpDesk / Add Usuario

POST http://localhost:8080/HelpDesk/api/usuario/add

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "id": null,
3   "nome": "Karina Almeida",
4   "email": "karina.almeida@fatec.br",
5   "senha": "karina1234",
6   "cargo": "gerente de projetos",
7   "telefone": "(11) 88785-2394",
8   "perfil": {
9     "id": 5,
10    "nomePerfil": "GERENTE"
11  }
12 }
```



Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
1 {
2   "id": 2,
3   "nome": "Karina Almeida",
4   "email": "karina.almeida@fatec.br",
5   "senha": "karina1234",
6   "cargo": "gerente de projetos",
7   "telefone": "(11) 88785-2394",
8   "perfil": {
9     "id": 5,
10    "nomePerfil": "GERENTE"
11  }
12 }
```



R – Read / SELECT

Listar Usuários

SERVICE

```
24     public List<Usuario> findAll() { 2 usages
25         return usuarioRepository.findAll();
26     }
```

- Os próprios recursos do JPA / Hibernate já fornece método que lista todos os registros de uma determinada entidade

MAPPER

```
58     public static List<UsuarioDTO> mapUsuariosToDTO(List<Usuario> usuarios) { 2 usages
59         if(usuarios != null && !usuarios.isEmpty()) {
60             List<UsuarioDTO> dto = new ArrayList<>();
61             usuarios.forEach(usuario -> dto.add(UsuarioDTO.valueOf(usuario)));
62             return dto;
63         }
64         return null;
65     }
```

- Ao invés de usar um for tradicional, é possível abreviar com forEach
- usuario é uma variável imutável, gerada a cada iteração do for
- O contexto lambda (→) são os trechos de código a serem executados a cada iteração

CONTROLLER

```
26      @GetMapping(🌐✓"/all")
27      public @ResponseBody ResponseEntity<List<UsuarioDTO>> all() {
28          List<Usuario> usuarios = usuarioService.findAll();
29          return ResponseEntity.ok().body(MapperGeral.mapUsuariosToDTO(usuarios));
30      }
```

- GetMapping : vai devolver para o cliente os valores pesquisados.
- Como se trata de uma lista de usuários, para facilitar o mapeamento do retorno via DTO, utiliza-se uma padronização no próprio Mapper.

TESTE

HTTP HelpDesk / All Usuario

GET http://localhost:8080/HelpDesk/api/usuario/all

Body Cookies Headers (5) Test Results 200 OK

{ } JSON Preview Visualize

```
1  [
2    {
3      "id": 2,
4      "nome": "Karina Almeida",
5      "email": "karina.almeida@fatec.br",
6      "senha": "karina4321",
7      "cargo": "gerente de projetos",
8      "telefone": "(11) 88785-2394",
9      "perfil": {
10       "id": 5,
11       "nomePerfil": "GERENTE"
12     }
13   }
14 ]
```

The top of the slide 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.

U – update

Atualizar senha

SERVICE

```
28     public Usuario findByEmail(String email) { 2 usages
29         if(email != null && !email.isEmpty()) {
30             return usuarioRepository.findByEmail(email).orElse( other: null);
31         }
32         return null;
33     }
```

- Será necessário buscar um usuário específico pelo e-mail, verificar se a senha antiga é igual a armazenada para só depois realizar a alteração.

SERVICE

```
35     public Usuario mudarSenha(String email, String senhaAntiga, String novaSenha) { 1 usage
36         Usuario usuario = findByEmail(email);
37         if (usuario != null) {
38             if(usuario.getSenha().equals(senhaAntiga)) {
39                 usuario.setSenha(novaSenha);
40                 return usuarioRepository.save(usuario);
41             }
42         }
43         return null;
44     }
```

- Será necessário buscar um usuário específico pelo e-mail, verificar se a senha antiga é igual a armazenada para só depois realizar a alteração.

CONTROLLER

```
32     @PutMapping("/mudarSenha/{email}/{senhaAntiga}/{senhaNova}")
33     public @ResponseBody ResponseEntity<UsuarioDTO> mudarSenha(@PathVariable String email,
34                                                                    @PathVariable String senhaAntiga,
35                                                                    @PathVariable String senhaNova) {
36         Usuario usuario = usuarioService.mudarSenha(email, senhaAntiga, senhaNova);
37         if(usuario != null)
38             return ResponseEntity.ok().body(UsuarioDTO.valueOf(usuario));
39         else
40             return ResponseEntity.badRequest().build();
41     }
```

- Para esse exemplo, vamos implementar a funcionalidade de atualização de senha.
- Se a mudança der certo, retorna o usuário em tela. Do contrário, o cliente recebe um erro 400 (badRequest), sinalizando que a operação não foi concretizada.

TESTE

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/HelpDesk/api/usuario/mudarSenha/karina.almeida@fatec.br/karina4321`
- Method:** `PUT`
- Status:** `200 OK` (220 ms, 351 B)
- Response Body (JSON):**

```
1  {
2    "id": 2,
3    "nome": "Karina Almeida",
4    "email": "karina.almeida@fatec.br",
5    "senha": "karina4321",
6    "cargo": "gerente de projetos",
7    "telefone": "(11) 88785-2394",
8    "perfil": {
9      "id": 5,
10     "nomePerfil": "GERENTE"
11   }
12 }
```

The top of the slide features a dark red header with a series of overlapping semi-circles. Each semi-circle contains a different pattern: concentric lines, a grid of dots, or a series of parallel diagonal lines.

D - delete


Remover usuário

SERVICE

```
46     public void deletarUsuario(Long id) { 1 usage
47         Usuario usuario = usuarioRepository.findById(id).orElse(other: null);
48         if(usuario != null) {
49             usuarioRepository.delete(usuario);
50         }
51     }
```

- Necessário verificar se o id informado, de fato, pertence a um usuário específico.
- Em caso positivo, procede com a remoção do usuário
- `findById()` : outro recurso pronto e facilitado com o uso do framework

SERVICE

```
46     public void deletarUsuario(Long id) { 1 usage
47          usuarioRepository.findById(id).ifPresent(usuario -> usuarioRepository.delete(usuario));
48     }
```

- A mesma implementação pode ser “resumida” em menos linhas de código
- `usuario` é uma variável imutável, reflexo do resultado de `findById()` e apenas existente no contexto dos parênteses do submétodo).

CONTROLLER

```
43     @DeleteMapping("/delete/{id}")
44     public @ResponseBody ResponseEntity<List<UsuarioDTO>> deleteUsuario(@PathVariable Long id) {
45         usuarioService.deletarUsuario(id);
46         List<Usuario> usuarios = usuarioService.findAll();
47         return ResponseEntity.ok().body(MapperGeral.mapUsuariosToDTO(usuarios));
48     }
```

- Lógica: a partir da lista de usuários, o cliente seleciona 1 específico para ser excluído (via id do usuário).

TESTE

- Via Postman, apague um usuário cadastrado.

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli

