PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli



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

```
@Table(name="usuario", schema = "public")
public class Usuario implements Serializable {
    @Serial
    public static final long serialVersionUID = -421308274897239279L;
    0Id
    @SequenceGenerator(
            name = "SEQ-USUARIO",
            sequenceName = "public.seg_usuario",
            allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ-USUARIO")
    private Long id;
    @Column(name = "nome", nullable = false, length = 255)
    private String nome;
    @Column(name = "email", nullable = false, length = 100)
    private String email;
    @Column(name = "senha", nullable = false, length = 255)
    private String senha;
    @Column(name = "cargo", nullable = false, length = 80)
    private String cargo;
```

```
@Column(name = "telefone", nullable = false, length = 30)
private String telefone;
@OneToOne
@JoinColumn(name = "id perfil", referencedColumnName = "id",
        insertable = true, updatable = true)
private Perfil perfil;
@OneToMany
@JoinTable(name = "usuario_eguipe", schema = "public",
        joinColumns = @JoinColumn(name = "id_usuario"),
        inverseJoinColumns = @JoinColumn(name = "id_equipe"))
@JsonManagedReference
private Collection<Equipe> equipes;
```

ENTIDADE USUÁRIO

```
• O QUE MUDA?
```

- 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.

```
public interface UsuarioRepository extends JpaRepository<Usuario, Long> { 2 usages

10

11

12

13

}
```

SERVIÇO

• Apenas o Serviço da entidade Usuario será criado

CONTROLLER

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

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.

```
public record UsuarioDTO( 3 usages

Long id, 1 usage
String nome, 1 usage
String email, 1 usage
String senha, 1 usage
String cargo, 1 usage
String telefone, 1 usage
PerfilDTO perfil 3 usages

implements Serializable {

OSerial no usages
public static final long serialVersionVID = 4021309213882379399L;
```

DTO USUÁRIO

```
public static UsuarioDTO valueOf(Usuario usuario) { no usages
   if (usuario != null) {
        return new UsuarioDTO(
                usuario.getId(),
                usuario.getNome(),
                usuario.getEmail(),
                usuario.getSenha(),
                usuario.getCargo(),
                usuario.getTelefone(),
                PerfilDTO.valueOf(usuario.getPerfil())
        );
   return null;
```

DTO USUÁRIO

```
public static Usuario toUsuario(UsuarioDTO dto) { 2 usages
40
               if (dto != null) {
                   Usuario usuario = new Usuario();
                   usuario.setId(dto.id);
                   usuario.setNome(dto.nome);
                   usuario.setEmail(dto.email);
                   usuario.setSenha(dto.senha);
                   usuario.setCargo(dto.cargo);
                   usuario.setTelefone(dto.telefone);
                   if (dto.perfil != null) {
                       usuario.setPerfil(PerfilDTO.toPerfil(dto.perfil));
                   else usuario.setPerfil(null);
                   return usuario;
               return null;
```

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

```
public Usuario cadastrar(Usuario usuario) { 2 usages

if (usuario != null) {

return usuarioRepository.save(usuario);

}

return null;

}
```

• 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"
```

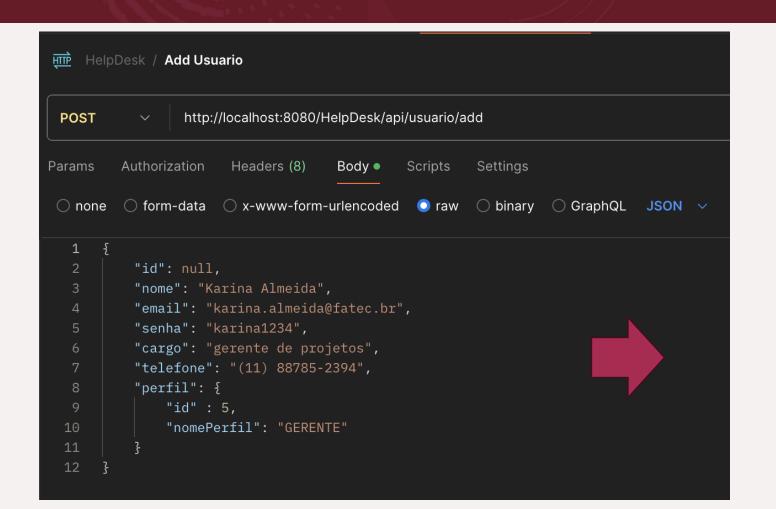
NA PRÁTICA

• 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



R - Read / SELECT

Listar Usuários

SERVICE

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

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

MAPPER

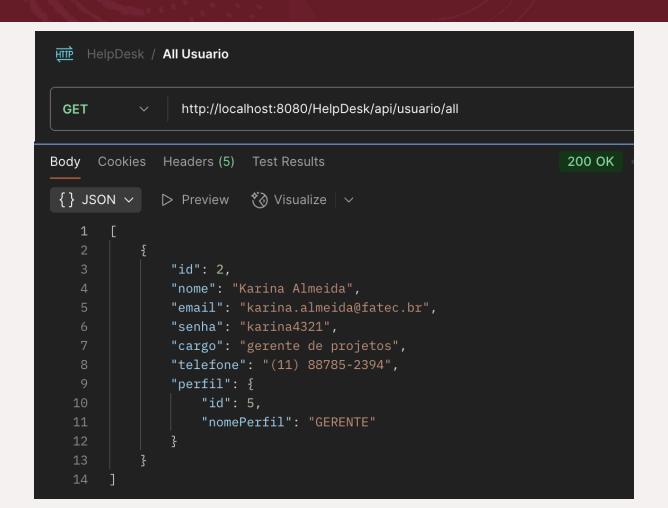
```
public static List<UsuarioDTO> mapUsuariosToDTO(List<Usuario> usuarios) { 2 usages
    if(usuarios != null && !usuarios.isEmpty()) {
        List<UsuarioDTO> dto = new ArrayList<>();
        usuarios.forEach(usuario -> dto.add(UsuarioDTO.valueOf(usuario)));
        return dto;
    }
    return null;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

- 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 (\rightarrow) são os trechos de código a serem executados a cada iteração

CONTROLLER

- 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



U - update

Atualizar senha

SERVICE

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

• 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

```
public Usuario mudarSenha(String email, String senhaAntiga, String novaSenha) { 1 usage
    Usuario usuario = findByEmail(email);
    if (usuario != null) {
        if(usuario.getSenha().equals(senhaAntiga)) {
            usuario.setSenha(novaSenha);
            return usuarioRepository.save(usuario);
    }
}
return null;
}
```

• 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

```
@PutMapping(@\*"/mudarSenha/{email}/{senhaAntiga}/{senhaNova}")

public @ResponseBody ResponseEntity<UsuarioDTO> mudarSenha(@PathVariable String email,

@PathVariable String senhaAntiga,

@PathVariable String senhaNova) {

Usuario usuario = usuarioService.mudarSenha(email, senhaAntiga, senhaNova);

if(usuario != null)

return ResponseEntity.ok().body(UsuarioDTO.valueOf(usuario));

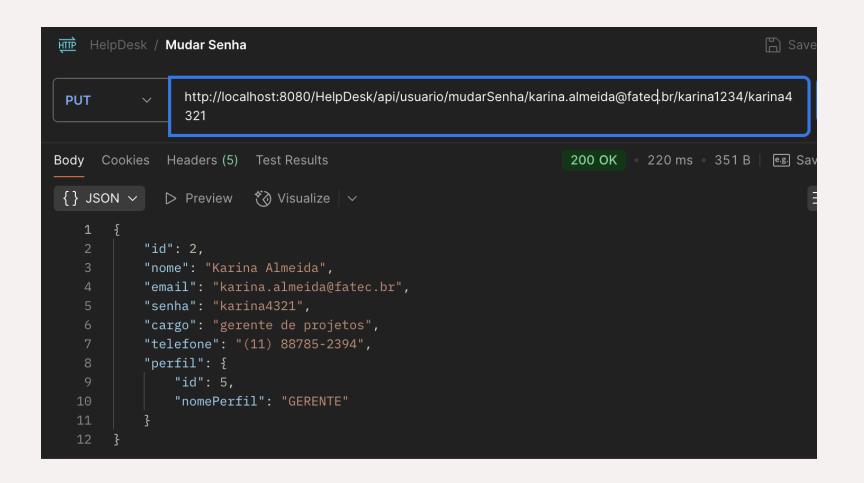
else

return ResponseEntity.badRequest().build();

}
```

- · 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



D - delete

Remover usuário

SERVICE

```
public void deletarUsuario(Long id) { 1usage

Usuario usuario = usuarioRepository.findById(id).orElse(other: null);

if(usuario != null) {

usuarioRepository.delete(usuario);
}
```

- 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

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

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

CONTROLLER

• 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

