# PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli

# CONTEXTO

Autenticação de Usuário

Aula 06

# CONTEXTO

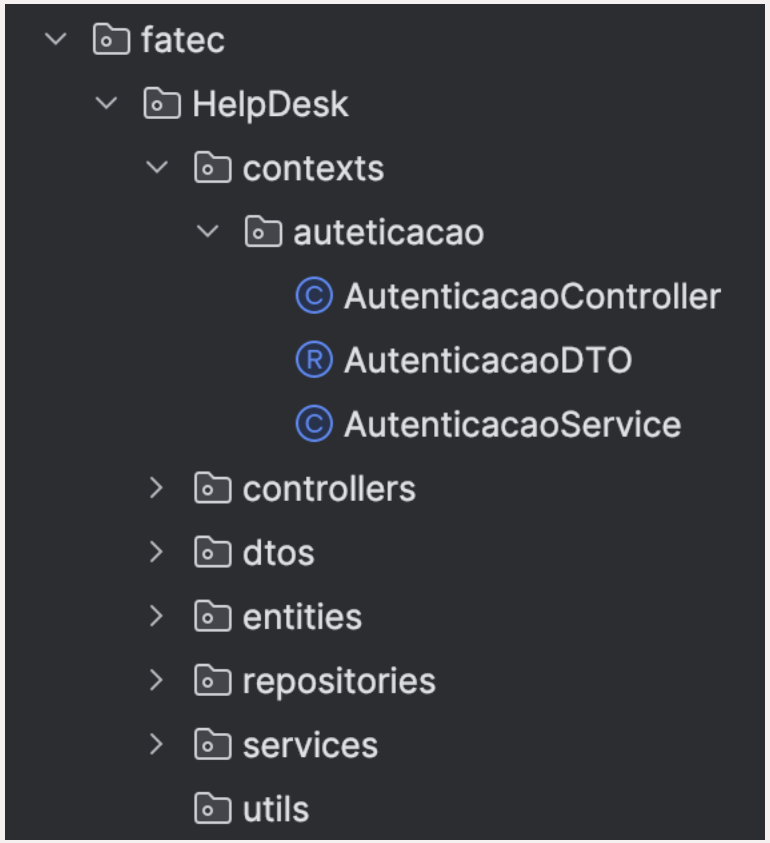
- Isolamento de funções específicas da API;
- Assim, as mudanças não interferem no andamento do projeto e nem em outras funcionalidades;

- Reutiliza código das classes principais
- Cria domínios exclusivos e desacoplados do núcleo do projeto

- A autenticação utiliza majoritariamente a tabela de Usuarios

# CRIANDO O CONTEXTO

- Pacote exclusivo dos contextos
- Dentro dos contextos, criar o pacote de Autenticação
- A autenticação inicial será básica: formada por verificação de e-mail e senha do usuário
- O retorno será um DTO com o nome, cargo, e-mail e perfil do usuário autenticado



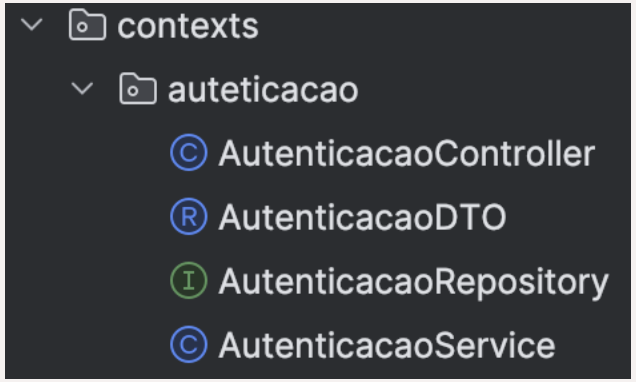
# AUTENTICAÇÃO

Primeiro Contexto

# AUTENTICAÇÃO

- Basicamente, recebe o e-mail e a senha do usuário.
- Busca o usuário que possui o e-mail específico e confere a senha
- Estando OK, devolve os dados do usuário

- Os elementos a seguir serão direcionados para a criação desse contexto com essa funcionalidade básica.



# DTO

- Os dados que queremos receber / informar para o cliente
- Note que IDs e senha não são revelados

```
public record AutenticacaoDTO(  8 usages

        String nome,  no usages
        String email,  no usages
        String cargo,  no usages
        String perfil  no usages

) implements Serializable {
```

```
    @Serial
    public static final long serialVersionUID = 3982284292937232983L;

    public static AutenticacaoDTO valueof(Usuario usuario) {  2 usages
        if (usuario != null) {
            return new AutenticacaoDTO(
                    usuario.getNome(),
                    usuario.getEmail(),
                    usuario.getCargo(),
                    usuario.getPerfil().getPerfil()
            );
        } else return null;
    }
```

# REPOSITORY

- Como trata de uma consulta específica, podemos implementar manualmente
- Toda consulta SQL fica nessa classe, separada da regra de negócio
- Nesse momento, estamos usando uma SQL Nativa

```
9  public interface AutenticacaoRepository extends JpaRepository<Usuario, Long> {  1 usage
10
11     @Query(value = "SELECT usu FROM Usuario usu " +  1 usage
12             "WHERE usu.email=:email AND usu.senha=:senha", nativeQuery = true)
13     Optional<Usuario> buscarUsuarioEmailSenhaSQL(String email, String senha);
14
15 }
```

# REPOSITORY / HQL

- Embora útil, a SQL Nativa possui pontos negativos, inclusive de otimização
- Usaremos no projeto o formato HQL
- HQL: a SQL é escrita com base no mapeamento das classes Java
- Nesse momento, o impacto / diferença é praticamente nula
- Em exemplos futuros, iremos notar as diferenças

```
9  public interface AutenticacaoRepository extends JpaRepository<Usuario, Long> {  1 usage
10
11     @Query(value = "SELECT usuario FROM Usuario usuario " +  1 usage
12             "WHERE usuario.email=:email AND usuario.senha=:senha", nativeQuery = false)
13     Optional<Usuario> buscarUsuarioEmailSenha(String email, String senha);
```

# SERVICE

- Implementa a regra de negócio de autenticação
- Nesse primeiro momento, usaremos a autenticação por SQL

```
10  @Service  1 usage
11  public class AutenticacaoService {
12
13      @Autowired
14      AutenticacaoRepository autenticacaoRepository;
15
16      public AutenticacaoDTO autenticarViaSQL(String email, String senha) {  no usages
17          Usuario usuario = autenticacaoRepository.buscarUsuarioEmailSenhaSQL(email, senha).orElse( other: null);
18          return AutenticacaoDTO.valueof(usuario);
19      }
20
21  }
```

# SERVICE

- O Service deve ser alterado para utilizar a HQL

```
11  public class AutenticacaoService {
12
13      @Autowired
14      AutenticacaoRepository autenticacaoRepository;
15
16      public AutenticacaoDTO autenticar(String email, String senha) {  1 usage
17          Usuario usuario = autenticacaoRepository.buscarUsuarioEmailSenha(email, senha).orElse( other: null);
18          return AutenticacaoDTO.valueof(usuario);
19      }
```

# CONTROLLER

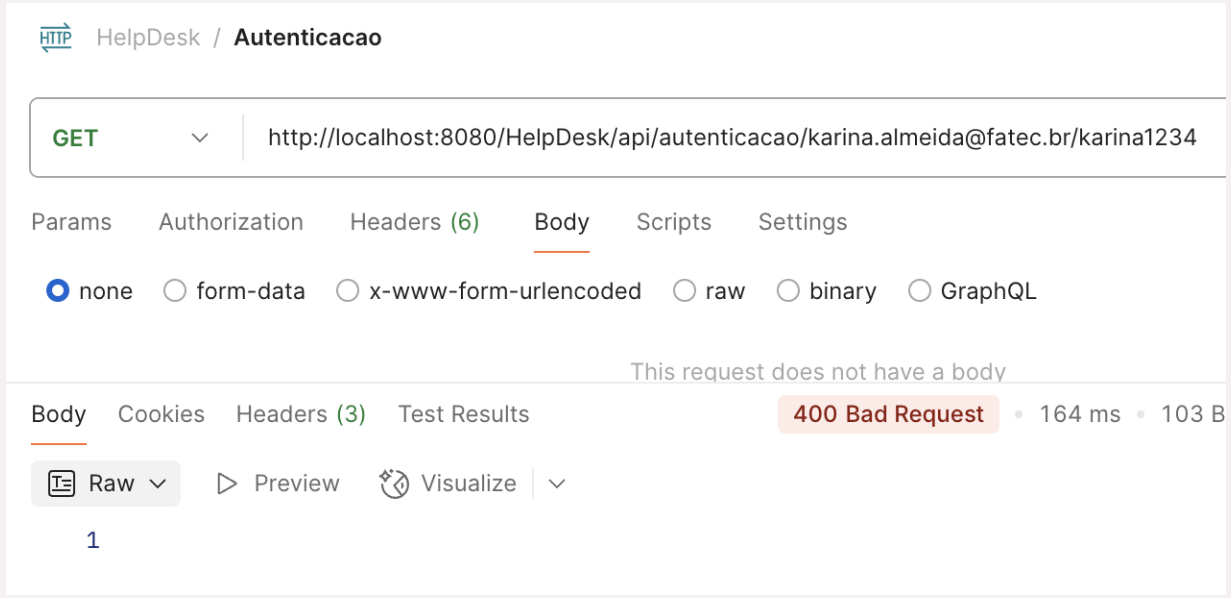
- Endpoint específico para realizar a autenticação dos clientes

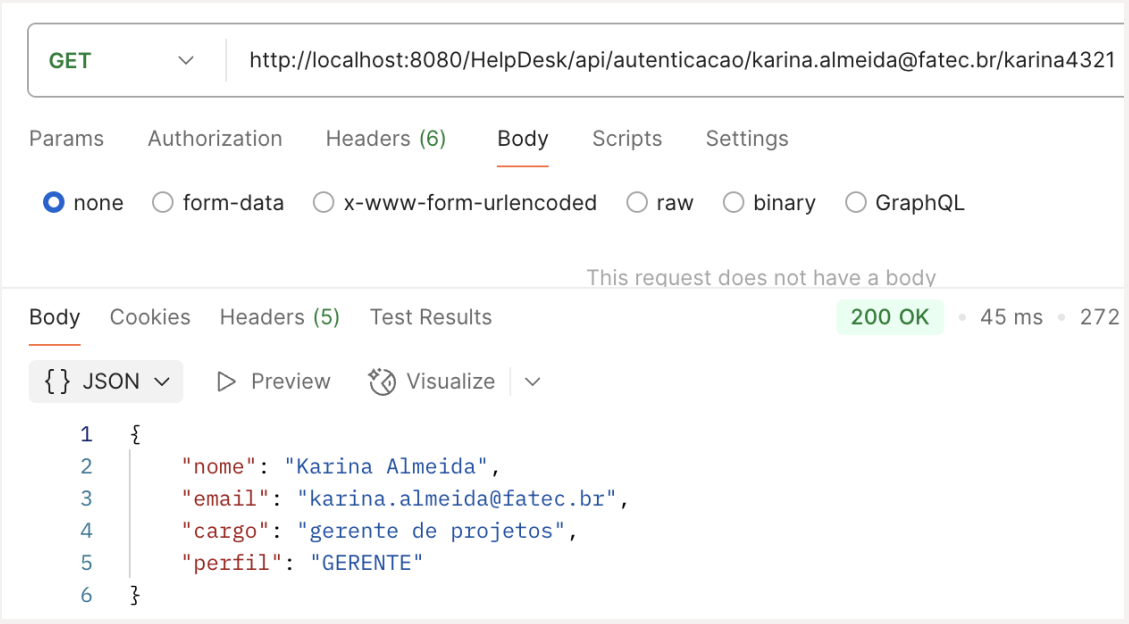
```java
@RestController
@RequestMapping("/autenticacao")
public class AutenticacaoController {

    @Autowired
    AutenticacaoService autenticacaoService;

    @GetMapping("/{email}/{senha}")
    public @ResponseBody ResponseEntity<AutenticacaoDTO> autenticar(@PathVariable String email,
                                                                    @PathVariable String senha) {
        AutenticacaoDTO autenticacaoDTO = autenticacaoService.autenticar(email, senha);
        if (autenticacaoDTO != null) {
            return ResponseEntity.ok().body(autenticacaoDTO);
        } else
            return ResponseEntity.badRequest().build();
    }

}
```

*Devolve um erro e não autoriza o usuário*

# TESTE

- Simulando o erro de autenticação e a autenticação satisfatória



# PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli