Java Spring Expert Capítulo 4

Casos de uso, signup, finalização

Competências

- Realização de caso de uso
    - Consulta detalhada de produtos
    - Signup
    - Recuperação de senha
    - Obter usuário logado
- Consultas ao banco dados
- Envio de email com Gmail

Material de revisão de SQL e ORM

Revisão Álgebra Relacional e SQL

Objetivo: relembrar as operações básicas com SQL.
https://www.youtube.com/watch?v=GHpE5xOxXXI

Super revisão de OO e SQL com Java e JDBC

Objetivo: compreender na prática como é consultar os dados de um banco de dados somente com Java e JDBC, sem utilizar uma ferramenta ORM (Mapeamento Objeto-Relacional).
https://www.youtube.com/watch?v=xC_yKw3MYX4

Nivelamento ORM - JPA e Hibernate

Objetivo: ter uma introdução teórica e prática sobre ORM com JPA, antes de ir direto para o Spring com o Spring Data JPA.
https://www.youtube.com/watch?v=CAP1IPgeJkw

Problema N+1 consultas relacionamento para-muitos

Objetivo: aprender como evitar idas e vindas desnecessárias ao banco de dados, quando fazemos uma consulta de entidades associadas para-muitos.
https://www.youtube.com/watch?v=sqbqoR-lMf8

Problema N+1 consultas relacionamento para-um

Objetivo: aprender como evitar idas e vindas desnecessárias ao banco de dados, quando fazemos uma consulta de entidades associadas para-um.
(próximo vídeo)

Caso de uso consulta paginada de produtos

Cenário principal:

1. [OUT] O sistema informa id e nome de todas categorias de produto
2. [IN] O usuário informa:
   - trecho do nome do produto (opcional)
   - categorias de produto desejadas (opcional)
   - número da página desejada
   - quantidade de itens por página
3. [OUT] O sistema informa uma listagem paginada dos produtos com suas respectivas categorias, conforme os critérios de consulta, ordenados por nome.

Protótipos de tela:

https://www.figma.com/file/cNa2l3TqZXxbU6NBDPruNw/BDS-DSCatalog

Planejando a requisição:

`/products?page=0&size=12&name=ma&categoryId=1,3`

Consultas para referência

```
@Query(nativeQuery = true, value = """
        SELECT DISTINCT tb_product.id, tb_product.name
        FROM tb_product
        INNER JOIN tb_product_category ON tb_product_category.product_id = tb_product.id
        WHERE (:categoryIds IS NULL OR tb_product_category.category_id IN (:categoryIds))
        AND (LOWER(tb_product.name) LIKE LOWER(CONCAT('%',:name,'%')))
        ORDER BY tb_product.name
        """,
        countQuery = """
        SELECT COUNT(*) FROM (
        SELECT DISTINCT tb_product.id, tb_product.name
        FROM tb_product
        INNER JOIN tb_product_category ON tb_product_category.product_id = tb_product.id
        WHERE (:categoryIds IS NULL OR tb_product_category.category_id IN (:categoryIds))
        AND (LOWER(tb_product.name) LIKE LOWER(CONCAT('%',:name,'%')))
        ) AS tb_result
        """)
Page<ProductProjection> searchProducts(List<Long> categoryIds, String name, Pageable pageable);

@Query("SELECT obj FROM Product obj JOIN FETCH obj.categories "
                + "WHERE obj.id IN :productIds ORDER BY obj.name")
List<Product> searchProductsWithCategories(List<Long> productIds);
```

Incluir Postgresql ao projeto no perfil dev

Dependência Maven

```
<dependency>
	<groupId>org.postgresql</groupId>
	<artifactId>postgresql</artifactId>
	<scope>runtime</scope>
</dependency>
```

Arquivo application-dev.properties

```
#spring.jpa.properties.jakarta.persistence.schema-generation.create-source=metadata
#spring.jpa.properties.jakarta.persistence.schema-generation.scripts.action=create
#spring.jpa.properties.jakarta.persistence.schema-generation.scripts.create-target=create.sql
#spring.jpa.properties.hibernate.hbm2ddl.delimiter=;

spring.datasource.url=jdbc:postgresql://localhost:5433/dscatalog
spring.datasource.username=postgres
spring.datasource.password=1234567

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.hibernate.ddl-auto=none
```

Executar Postgresql e pgAdmin (ou DBeaver)

Opção 1: instalar diretamente o Postgresql e o pgAdmin ou DBeaver no seu sistema

Opção 2: subir Postgresql e pgAdmin via Docker Compose:
https://gist.github.com/acenelio/5e40b27cfc40151e36beec1e27c4ff71

Dica: gerar comandos SQL para excluir tabelas

```
SELECT 'drop table if exists ' || tablename || ' cascade;'
FROM pg_tables
WHERE schemaname = 'public';
```

Caso de uso sign up

Cenário principal:

1. [IN] O usuário informa primeiro nome, sobrenome, email e senha

Exceção 1.1: Erro de validação

1.1.1. [OUT] O sistema informa os erros de validação

Informações complementares

Critérios de validação de usuário

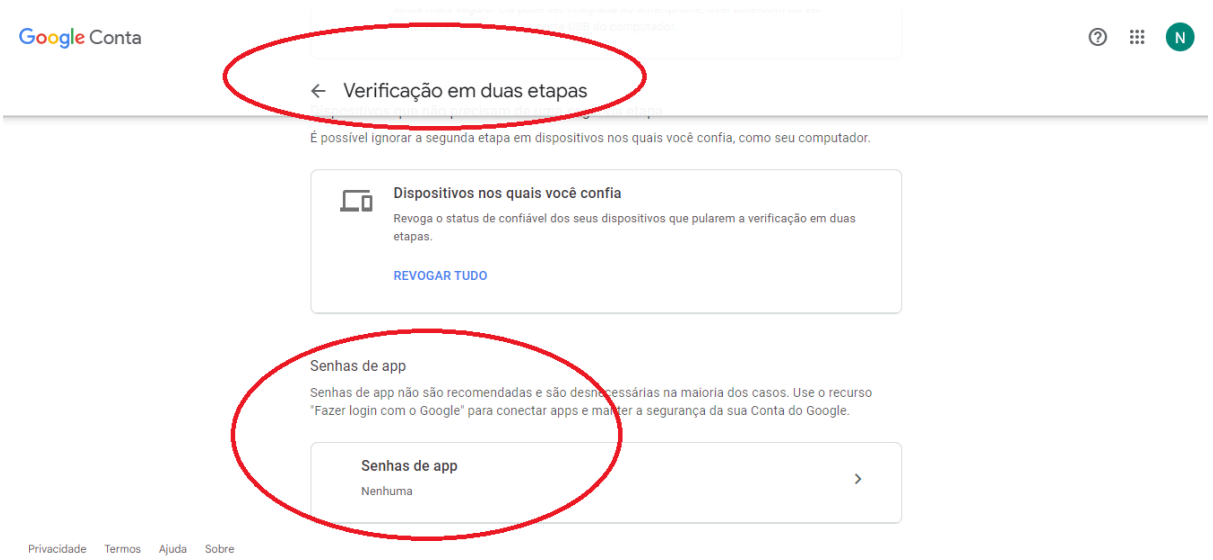
- Nome: campo requerido
- Email: email válido
- Senha: mínimo 8 caracteres

Envio de email via Gmail

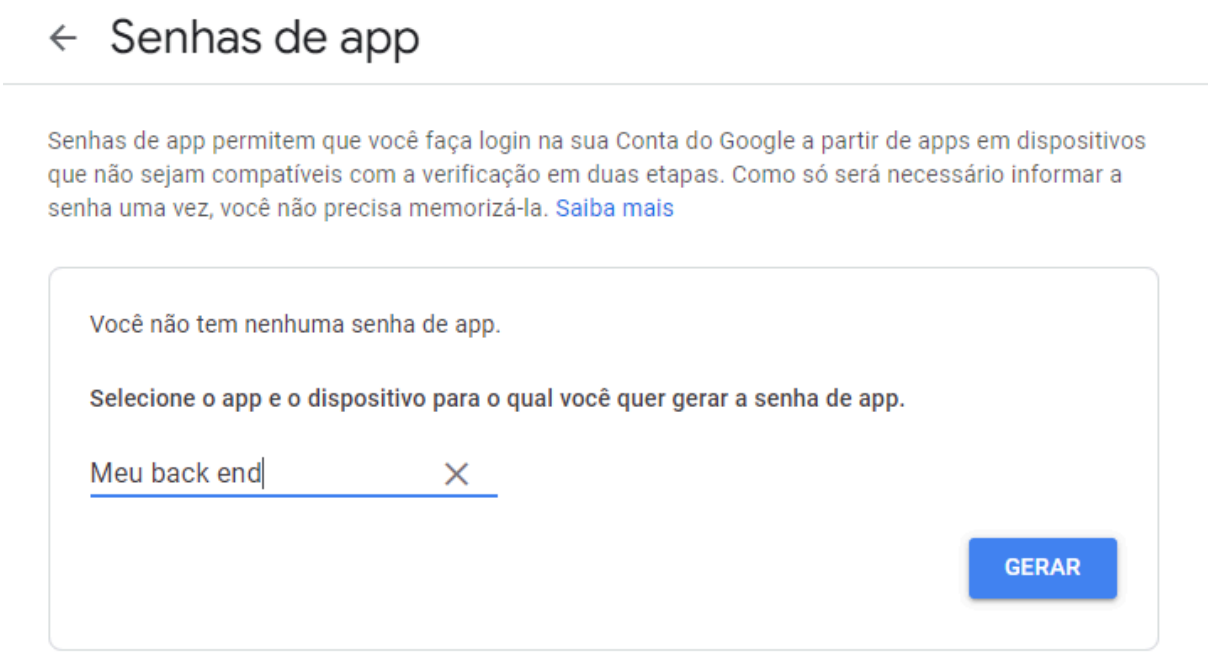
1. Criar uma senha de app na sua conta do Google

Guia:
https://support.google.com/accounts/answer/185833

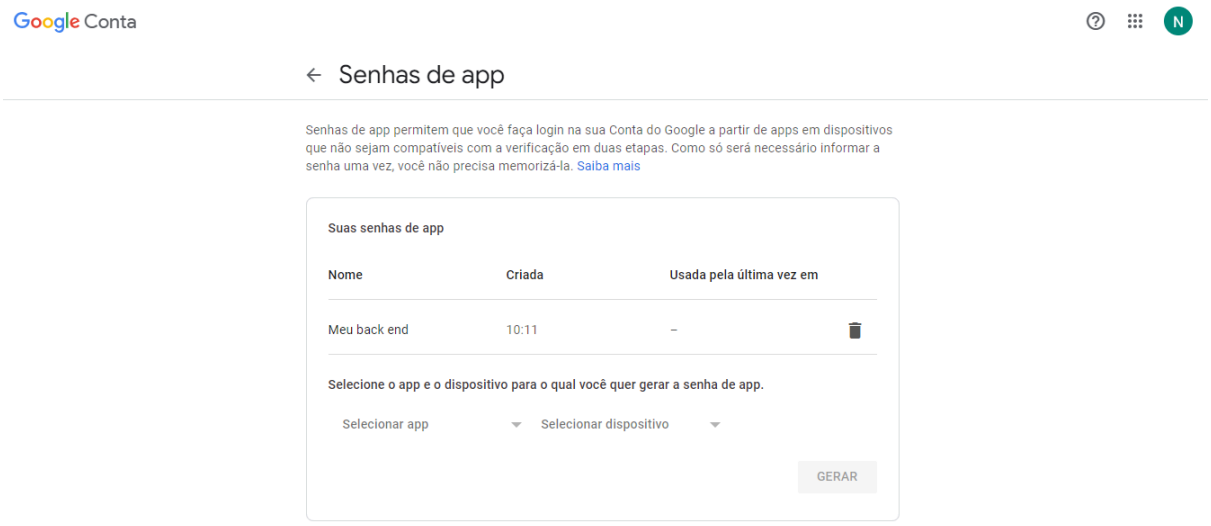
Acessar sua conta no Google -> Segurança -> Validação em duas etapas -> Senhas de app
https://myaccount.google.com/



Selecionar dispositivo -> Outro -> (escolha um nome que lembre seu app) -> Gerar



Pronto! Sua senha de app de 16 caracteres foi gerada. Salve-a em um lugar seguro.



2. Abra o projeto referência na sua IDE:

https://github.com/devsuperior/spring-boot-gmail

3. Configure as variáveis de ambiente EMAIL_USERNAME e EMAIL_PASSWORD

* Essas variáveis estão definidas no arquivo application.properties

ATENÇÃO: nunca escreva suas credenciais diretamente no application.properties. Configure os valores das variáveis no ambiente de execução do projeto.

4. Execute o projeto e teste a requisição de envio de email

POST http://localhost:8080/email

Corpo da requisição:

```
{
    "to": "destinatario@gmail.com",
    "subject": "Aviso aos clientes",
    "body": "Prezado cliente,\n\nAcesse agora:\n\nhttps://devsuperior.com.br\n\nAbraços!"
}
```

Caso de uso recuperação de senha

Cenário principal:

1. [IN] O usuário informa seu email
2. [OUT] O sistema informa o token de recuperação e a validade do mesmo
3. [IN] O usuário informa o token de recuperação e a nova senha

Exceção 1.1: Email inválido

1.1.1. [OUT] O sistema informa que o email é inválido

Exceção 1.2: Email não encontrado

1.2.1. [OUT] O sistema informa que o email não foi encontrado

Exceção 3.1: Token inválido

3.1.1. [OUT] O sistema informa que o token é inválido

Exceção 3.2: Erro de validação

3.1.2. [OUT] O sistema informa que a senha é inválida

Informações complementares

Critérios de validação de senha:

- Mínimo 8 caracteres

Variáveis para recuperação de senha

```
email.password-recover.token.minutes=${PASSWORD_RECOVER_TOKEN_MINUTES:30}
email.password-recover.uri=${PASSWORD_RECOVER_URI:http://localhost:5173/recover-password/}
```

Consulta para encontrar o token não expirado

```
@Query("SELECT obj FROM PasswordRecover obj WHERE obj.token = :token AND obj.expiration > :now")
List<PasswordRecover> searchValidTokens(String token, Instant now);
```

Obter usuário logado

```
protected User authenticated() {
  try {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Jwt jwtPrincipal = (Jwt) authentication.getPrincipal();
    String username = jwtPrincipal.getClaim("username");
    return userRepository.findByEmail(username);
  }
  catch (Exception e) {
    throw new UsernameNotFoundException("Invalid user");
  }
}
```