


 [devsuperior](#) / [sds-dsmovie](#) Public[Code](#) [Issues 1](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [main](#) ▾

...

[sds-dsmovie](#) / [episodio2](#) /

acenelio ...

5 hours ago 

..



README.md

5 hours ago

 README.md

## Semana Spring React - Episódio 2

*Crie um app inédito para seu portfólio com as tecnologias mais demandadas do mercado*

### Realização

[DevSuperior - Escola de programação](#)

### Objetivos do projeto para esta aula

- Implementar o back end
- Modelo de domínio
- Acesso a banco de dados
- Estruturar o back end no padrão camadas
- Criar endpoints da API REST
- Implantação na nuvem

**AVISO:** as aulas ficarão disponíveis somente até domingo às 23h59

# AVISO: Instruções sobre certificado no Github do [aqui](#)

## Checklist

### Passo: configuração de segurança

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private Environment env;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        if (Arrays.asList(env.getActiveProfiles()).contains("test")) {
            http.headers().frameOptions().disable();
        }

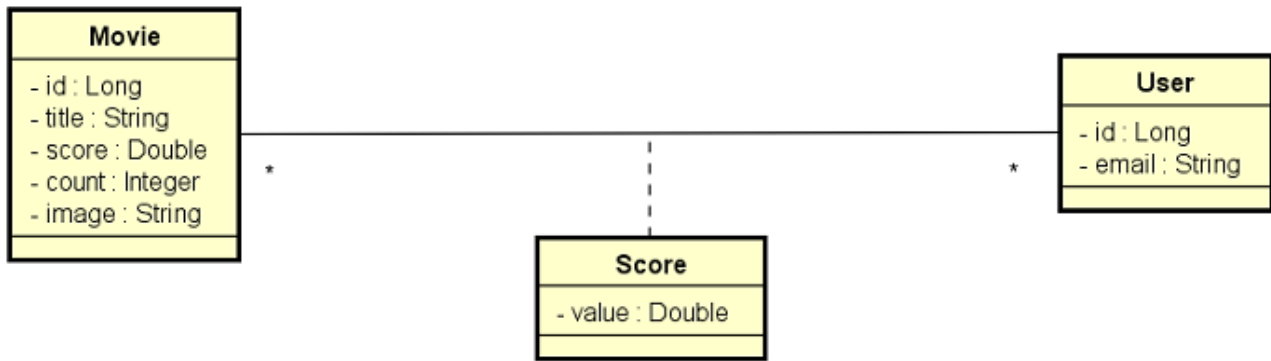
        http.cors().and().csrf().disable();
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.
            http.authorizeRequests().anyRequest().permitAll();
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration().applyPermit
            configuration.setAllowedMethods(Arrays.asList("POST", "GET", "PUT", "
        final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfig
            source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

- COMMIT: Security config

### Passo: criar as entidades e o seed do banco

#### Modelo conceitual



## application.properties

```

spring.profiles.active=test

spring.jpa.open-in-view=false

```

## application-test.properties

```

# Dados de conexão com o banco H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

# Configuração do cliente web do banco H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Configuração para mostrar o SQL no console
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

```

## import.sql

```

INSERT INTO tb_user(email) VALUES ('maria@gmail.com');
INSERT INTO tb_user(email) VALUES ('joao@gmail.com');
INSERT INTO tb_user(email) VALUES ('ana@gmail.com');
INSERT INTO tb_user(email) VALUES ('lucia@gmail.com');
INSERT INTO tb_user(email) VALUES ('joaquim@gmail.com');

INSERT INTO tb_movie(score, count, title, image) VALUES (4.5, 2, 'The Witcher', 'http
INSERT INTO tb_movie(score, count, title, image) VALUES (3.3, 3, 'Venom: Tempo de Car
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'O Espetacular Homem-A
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Matrix Resurrections'
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Shang-Chi e a Lenda d
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Django Livre', 'https
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Titanic', 'https://ww

```

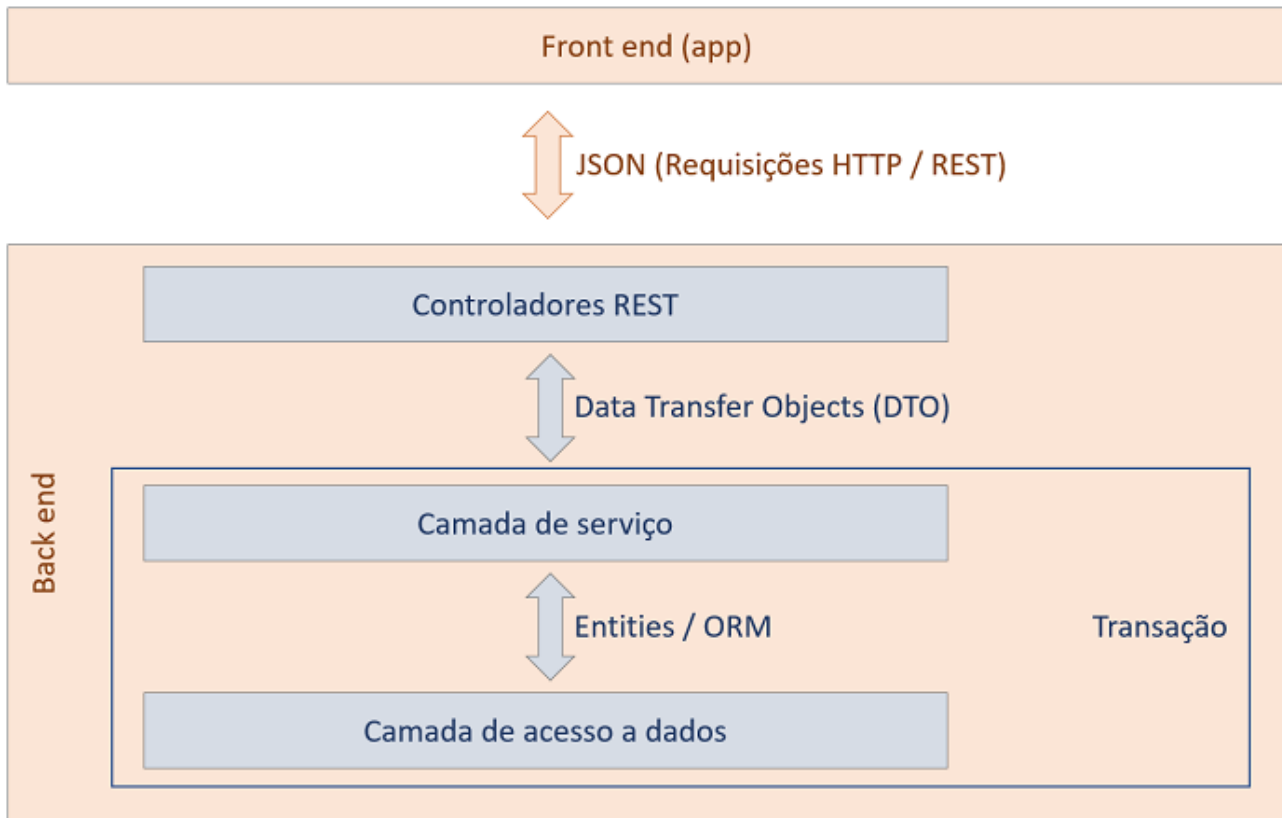
```
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'O Lobo de Wall Street
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Aves de Rapina: Arleq
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Rogue One: Uma Histór
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Star Wars: A Guerra d
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Star Wars: Episódio I
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Vingadores: Ultimato'
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Thor', 'https://www.t
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Cisne Negro', 'https:
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'O Silêncio dos Inocen
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Clube da Luta', 'http
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Guerra Mundial Z', 'h
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Harry Potter e as Rel
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Harry Potter e a Pedr
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Alice no País das Mar
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Animais Fantásticos e
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'A Teoria de Tudo', 'h
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'O Livro de Boba Fett'
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'O Último Duelo', 'htt
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Interestelar', 'https
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Contato', 'https://ww
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Duna', 'https://www.t
INSERT INTO tb_movie(score, count, title, image) VALUES (0, 0, 'Aquaman', 'https://ww

INSERT INTO tb_score(movie_id, user_id, value) VALUES (1, 1, 5.0);
INSERT INTO tb_score(movie_id, user_id, value) VALUES (1, 2, 4.0);
INSERT INTO tb_score(movie_id, user_id, value) VALUES (2, 1, 3.0);
INSERT INTO tb_score(movie_id, user_id, value) VALUES (2, 2, 3.0);
INSERT INTO tb_score(movie_id, user_id, value) VALUES (2, 3, 4.0);
```

- COMMIT: Domain model, database seed

## Passo: Busca de filmes

### Padrão camadas adotado



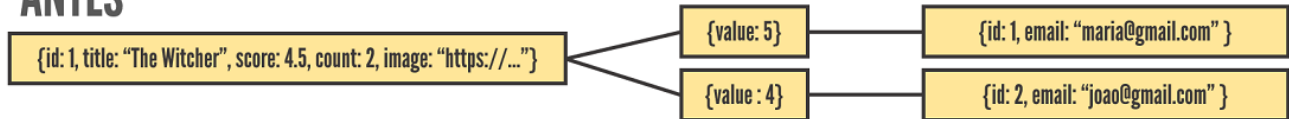
- Criar repository
- Criar DTO
- Criar service
- Criar controller
- COMMIT: Find movies

## Passo: Salvar avaliação

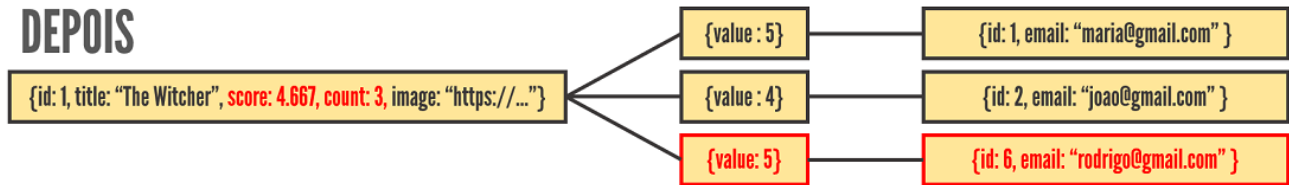
### Lógica:

1. Informar email, id do filme e valor da avaliação (1 a 5).
2. Recuperar usuário do banco de dados pelo email. Se o usuário não existir, insira no banco.
3. Salvar a avaliação do usuário para o dado filme.
4. Recalcular a avaliação média do filme e salvar no banco de dados.

## ANTES



## DEPOIS



- COMMIT: Save score

## Passo: Validação no Postgres local

- Criar três perfis de projeto: test, dev, prod
- Gerar script SQL no perfil dev
- Testar projeto no banco Postgres local

### application-dev.properties

```

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

spring.datasource.url=jdbc:postgresql://localhost:5432/dsmovie
spring.datasource.username=postgres
spring.datasource.password=1234567

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.hibernate.ddl-auto=none
  
```

### application-prod.properties

```
spring.datasource.url=${DATABASE_URL}
```

### system.properties

```
java.runtime.version=17
```

- COMMIT: First homolog

## Passo: Implantação no Heroku

- Criar app no Heroku
- Provisionar banco Postgres
- Definir variável APP\_PROFILE=prod
- Conectar ao banco via pgAdmin
- Criar seed do banco

```
heroku -v
heroku login
heroku git:remote -a <nome-do-app>
git remote -v
git subtree push --prefix backend heroku main
```

## Passo: implantação no Netlify

- Deploy básico
  - Base directory: frontend
  - Build command: yarn build
  - Publish directory: frontend/build
- Arquivo \_redirects

```
/* /index.html 200
```

- Configurações adicionais
  - Site settings -> Domain Management: (colocar o nome que você quiser)
  - Deploys -> Trigger deploy

## PARABÉNS!

---



- Quero muito saber seu feedback
  - O que você está achando da nossa abordagem?
  - Você está conseguindo acompanhar?
  - O que você está achando do evento?
- Participe

- Comente na página da Semana Spring React
  - Divulgue seu projeto no LinkedIn e marque a DevSuperior
-