

# tryber / sd-03-block13-project-movie-card-library-stateful

generated from [betrybe/sd-0x-project-movie-cards-library-stateful](#)

☆ 1 star    🔗 1 fork

☆ Star

🔔 Notifications

<> Code

! Issues

🔗 Pull requests 55

▶ Actions

📁 Projects

🛡 Security



🔗 master ▾

Go to file



jeanpsv Merge pull request #1 from tryber/feature/configure-produ...



✖ on 15 Apr 2020

🕒 3

[View code](#)

☰ README.md

## Boas vindas ao repositório do projeto de Movie Cards Library Stateful!

Você já usa o GitHub diariamente para desenvolver os exercícios, certo? Agora, para desenvolver os projetos, você deverá seguir as instruções a seguir. Fique atento a cada passo, e se tiver qualquer dúvida, nos envie por *Slack!* #vqv 🚀

Aqui você vai encontrar os detalhes de como estruturar o desenvolvimento do seu projeto a partir deste repositório, utilizando uma branch específica e um *Pull Request* para colocar seus códigos.

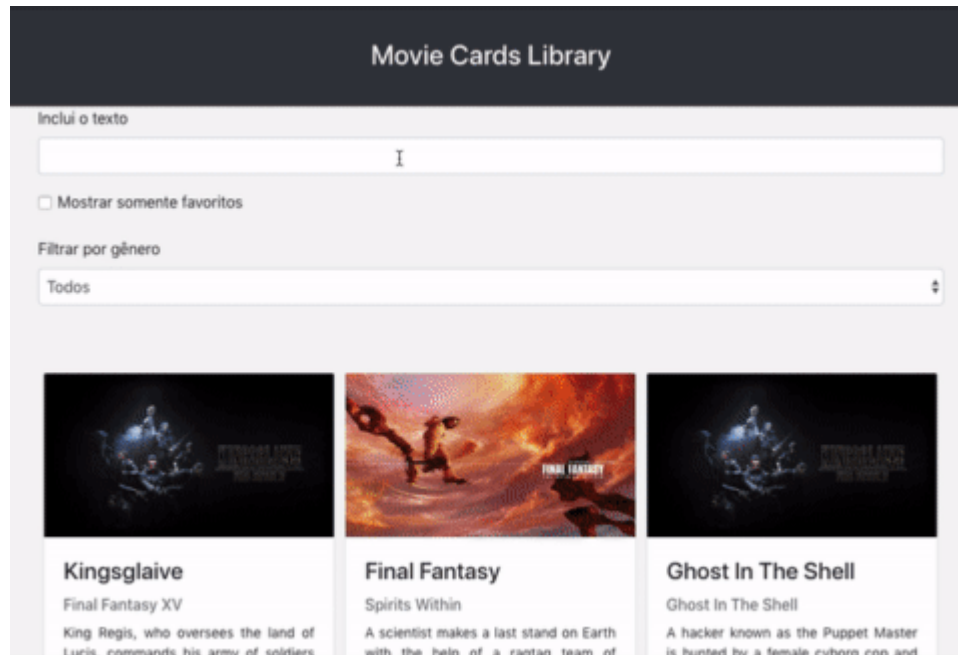
## O que deverá ser desenvolvido

Você deverá desenvolver uma aplicação que consiste em uma biblioteca de cartões de filmes dinâmica utilizando React. A biblioteca é composta por:

- Um cabeçalho;
- Uma barra de busca, utilizada pra filtrar quais cartões serão exibidos na lista de cartões;

- Uma lista de cartões, onde cada cartão representa um filme e possui uma imagem, título, subtítulo, sinopse e avaliação;
- Um formulário para adicionar um novo cartão na biblioteca.

Uma possível implementação dessa biblioteca consta abaixo.



Você precisará implementar componentes que em conjunto resultarão na biblioteca de cartões de filmes dinâmica.

## Desenvolvimento

Este repositório já contém um *template* com um App React criado e configurado. Após clonar o projeto e instalar as dependências (mais sobre isso abaixo), você não precisará realizar nenhuma configuração adicional. Você deverá completar este *template* implementando os requisitos listados na próxima seção.

## Componentes

Esse projeto contém os seguintes `React Components` :

- Header
- MovieLibrary
- SearchBar
- MovieList
- MovieCard

- Rating
- AddMovie

Header , MovieList , MovieCard e Rating já estão implementados. MovieLibrary , searchBar e AddMovie já estão criados, mas precisam ser implementados de forma a passarem nos requisitos listados abaixo.

Todos os componentes devem ser criados dentro da pasta src/components . É **obrigatório que seus componentes tenham exatamente os nomes listados acima.**

## Estado de componentes

Os componentes que **precisam** ter estado a ser gerenciado são:

- MovieLibrary
- AddMovie

## Testes

---

Todos os requisitos do projeto serão testados **automaticamente**. Cada componente possui vários requisitos. Os testes para cada requisito de um componente estão no arquivo de teste correspondente.

**Por exemplo:** Os requisitos relacionados ao componente searchBar estão no arquivo searchBar.test.js .

Separe um tempo para ler estes arquivos e entender como os testes estão organizados.

Para executar os testes localmente, digite no terminal o comando npm test .  
Inicialmente, seus testes estarão assim:



```

FAIL src/App.test.js
  <Header /> component
    ✕ renders without crashing (2ms)
    ✕ it includes the text `Movie Cards Library` inside a h1 tag
  <MovieList /> component
    ✕ renders without crashing
    ✕ renders a `MovieCard` component for each object in the array
    ✕ sets the movie title as the key in each rendered `MovieCard` (1ms)
  <MovieCard /> component
    ✕ renders without crashing
    ✕ renders the movie image inside an `image` tag
    ✕ renders the movie title inside an `h4` tag
    ✕ renders the movie subtitle inside an `h5` tag
    ✕ renders the movie storyline inside a `p` tag (1ms)
    ✕ renders a `Rating` component
    ✕ passes the rating attribute to the `Rating` component
  <Rating /> component
    ✕ renders without crashing (1ms)
    ✕ renders the rating inside an element with the class `rating`
  <App /> component
    ✕ renders a `Header` component (9ms)
    ✕ renders a `MovieList` component (1ms)
  • <Header /> component › renders without crashing

ReferenceError: Header is not defined

   36 | describe('<Header /> component', () => {
   37 |   it('renders without crashing', () => {
>  38 |     shallow(<Header />);
      |             ^
   39 |   });
   40 |
   41 |   it('it includes the text `Movie Cards Library` inside a h1 tag', () => {
      |
      at Object.<anonymous> (src/App.test.js:38:14)
  • <Header /> component › it includes the text `Movie Cards Library` inside a h1 tag

```

A primeira parte da saída mostra um sumário de cada teste e seu status. Um  representa um teste falhando, enquanto um  representa um teste correto. Naturalmente, no início todos os testes estarão falhando.

Abaixo do sumário, para cada teste falhando, há uma mensagem explicativa sobre o motivo que causou a falha do teste, assim como a linha em que a falha ocorreu. Na imagem, vemos que o teste falha porque o componente `Header`, utilizado na linha 38, não está definido.

Se fizermos uma implementação simples do componente `Header`, que não renderiza nada:

```

import React from 'react';

class Header extends React.Component {
  render() {
  }
}

```

```
export default Header;
```

E descomentarmos a linha que importa o componente `Header` em `App.test.js`:

```
// import App from './App';
import Header from './components/Header';
// import MovieCard from './components/MovieCard';
// import MovieList from './components/MovieList';
// import Rating from './components/Rating';
```

Veremos que o primeiro teste agora passa:

```
FAIL src/App.test.js
<Header /> component
  ✓ renders without crashing (4ms)
  ✗ it includes the text `Movie Cards Library` inside a h1 tag (37ms)
<MovieList /> component
  ✗ renders without crashing
  ✗ renders a `MovieCard` component for each object in the array
  ✗ sets the movie title as the key in each rendered `MovieCard` (1ms)
<MovieCard /> component
  ✗ renders without crashing
  ✗ renders the movie image inside an `image` tag
  ✗ renders the movie title inside an `h4` tag (1ms)
  ✗ renders the movie subtitle inside an `h5` tag
  ✗ renders the movie storyline inside a `p` tag
  ✗ renders a `Rating` component (1ms)
  ✗ passes the rating attribute to the `Rating` component
<Rating /> component
  ✗ renders without crashing
  ✗ renders the rating inside an element with the class `rating` (1ms)
<App /> component
  ✗ renders a `Header` component (4ms)
  ✗ renders a `MovieList` component (1ms)

• <Header /> component ✗ it includes the text `Movie Cards Library` inside a h1 tag

Method "text" is meant to be run on 1 node. 0 found instead.

   42 |         wrapper = shallow(<Header />);
   43 |
>  44 |         expect(wrapper.find('header h1').text()).toBe('Movie Cards Library');
      |                                     ^
   45 |     });
   46 | });
   47 |

at ShallowWrapper.single (node_modules/enzyme/src/ShallowWrapper.js:1636:13)
at ShallowWrapper.text (node_modules/enzyme/src/ShallowWrapper.js:1079:17)
at Object.<anonymous> (src/App.test.js:44:38)
```

Quando seu projeto estiver terminado, todos os testes deverão estar passando:

```
PASS src/App.test.js
  <Header /> component
    ✓ renders without crashing (4ms)
    ✓ it includes the text `Movie Cards Library` inside a h1 tag (4ms)
  <MovieList /> component
    ✓ renders without crashing
    ✓ renders a `MovieCard` component for each object in the array (13ms)
    ✓ sets the movie title as the key in each rendered `MovieCard` (36ms)
  <MovieCard /> component
    ✓ renders without crashing (1ms)
    ✓ renders the movie image inside an `image` tag (1ms)
    ✓ renders the movie title inside an `h4` tag (1ms)
    ✓ renders the movie subtitle inside an `h5` tag (1ms)
    ✓ renders the movie storyline inside a `p` tag (1ms)
    ✓ renders a `Rating` component (1ms)
    ✓ passes the rating attribute to the `Rating` component (13ms)
  <Rating /> component
    ✓ renders without crashing
    ✓ renders the rating inside an element with the class `rating` (1ms)
  <App /> component
    ✓ renders a `Header` component (2ms)
    ✓ renders a `MovieList` component (1ms)
```

```
Test Suites: 1 passed, 1 total
Tests:       16 passed, 16 total
Snapshots:   0 total
Time:        2.645s, estimated 3s
Ran all test suites related to changed files.
```

Watch Usage: Press w to show more. ■

## Dica: desativando testes

Especialmente no início, quando a maioria dos testes está falhando, a saída após executar os testes é bastante poluída. Você pode desabilitar temporariamente um teste utilizando a função `skip` junto à função `it`. Como o nome indica, esta função "pula" um teste:

```
it.skip('it includes the text `Movie Cards Library` inside a h1 tag', () => {
  wrapper = shallow(<Header />);

  expect(wrapper.find('header h1').text()).toBe('Movie Cards Library');
});
```

Na saída da execução dos testes, você verá um ○ indicando que o teste está sendo pulado:

```
FAIL src/App.test.js
  <Header /> component
    ✓ renders without crashing (4ms)
    ○ skipped it includes the text `Movie Cards Library` inside a h1 tag
  <MovieList /> component
    ✗ renders without crashing (1ms)
    ✗ renders a `MovieCard` component for each object in the array (1ms)
    ✗ sets the movie title as the key in each rendered `MovieCard`
  <MovieCard /> component
    ✗ renders without crashing
    ✗ renders the movie image inside an `image` tag (1ms)
    ✗ renders the movie title inside an `h4` tag
    ✗ renders the movie subtitle inside an `h5` tag
    ✗ renders the movie storyline inside a `p` tag (1ms)
    ✗ renders a `Rating` component
    ✗ passes the rating attribute to the `Rating` component (1ms)
  <Rating /> component
    ✗ renders without crashing
    ✗ renders the rating inside an element with the class `rating`
  <App /> component
    ✗ renders a `Header` component (7ms)
    ✗ renders a `MovieList` component (1ms)
```

Uma estratégia é pular todos os testes no início e ir implementando um teste de cada vez, removendo dele a função `skip`.

⚠️ Lembre-se de não entregar o projeto com nenhum teste ignorado. **Testes ignorados serão tratados como testes falhando.** ⚠️

⚠️ Não apague, em hipótese alguma, qualquer teste ou arquivo deste repositório..  
⚠️

### Dica: watch mode

Ao executar os testes localmente, [Jest](#), a ferramenta que executa os testes, entra em *watch mode*. Nesse modo, a cada vez que um arquivo é salvo, os testes são executados novamente. Isso pode aumentar sua produtividade removendo a necessidade de executar os testes manualmente o tempo todo. Você pode abrir uma aba no seu terminal ou no terminal do VSCode e deixar o *Jest* rodando nesse modo.

## Requisitos do projeto

⚠️ Lembre-se que o seu projeto só será avaliado se estiver passando por **todos os checks** do **CodeClimate**. Use o *check* do **TravisCI** para medir o seu progresso em relação aos requisitos! 😊

Crie um componente chamado `SearchBar`

Esse componente renderizará uma barra com filtros acima da listagem de cartões. Quais cartões serão mostrados no componente `MovieList` dependerá dos filtros escolhidos.

`SearchBar` deve receber como props:

- `searchText` , uma string
- `onSearchTextChange` , uma callback
- `bookmarkedOnly` , um boolean
- `onBookmarkedChange` , uma callback
- `selectedGenre` , uma string
- `onSelectedGenreChange` , uma callback

### Renderize um formulário dentro de `SearchBar`

Dentro desse formulário haverá campos usados na filtragem de cartões.

#### Renderize um input do tipo texto dentro do formulário em `SearchBar`

- O input deve ter uma label associada com o texto: **"Inclui o texto:"**;
- A propriedade `value` do input deve receber o valor da prop `searchText` ;
- A propriedade `onChange` do input deve receber o valor da prop `onSearchTextChange` .

#### Renderize um input do tipo checkbox dentro do formulário em `SearchBar`

- O input deve ter uma label associada com o texto: **"Mostrar somente favoritos"**;
- A propriedade `checked` do input deve receber o valor da prop `bookmarkedOnly` ;
- A propriedade `onChange` do input deve receber o valor da prop `onBookmarkedChange` .

#### Renderize um select dentro do formulário em `SearchBar`

- O select deve ter uma label associada com o texto: **"Filtrar por gênero"**;
- A propriedade `value` do select deve receber o valor da prop `selectedGenre` ;
- A propriedade `onChange` do input deve receber o valor da prop `onSelectedGenreChange` ;
- O `select` deve renderizar quatro tags `option` , com as opções de filtragem por gênero, na seguinte ordem:



- `Todos` , com o valor `""` ;
- `Ação` , com o valor `action` ;
- `Comédia` , com o valor `comedy` ;
- `Suspense` , com o valor `thriller` .

## Crie um componente chamado `AddMovie`

Esse componente renderizará um formulário que permite adicionar na biblioteca um novo cartão de filme, dadas as seguintes informações do novo filme:

- subtítulo
- título
- caminho da imagem
- sinopse
- avaliação
- gênero

`AddMovie` deve receber como props:

- `onClick` , uma callback

## Configure o estado inicial do componente `AddMovie`

O componente `AddMovie` possui como estado as seguintes propriedades:

- `subtitle` : guarda o subtítulo preenchido no formulário por quem usa a aplicação;
- `title` : guarda o título preenchido no formulário por quem usa a aplicação;
- `imagePath` : guarda o caminho da imagem preenchido no formulário por quem usa a aplicação;
- `storyline` : guarda a sinopse do filme escrita no formulário por quem usa a aplicação;
- `rating` : guarda a nota de avaliação dada no formulário por quem usa a aplicação;
- `genre` : guarda o gênero do filme selecionado no formulário por quem usa a aplicação.

Ou seja, o estado de `AddMovie` contém as informações do novo filme que foram inseridas por quem usa a aplicação.

O estado inicial do componente `AddMovie` deve ser:

- `subtitle` : "";

- `title : '';`
- `imagePath : '';`
- `storyline : '';`
- `rating : 0;`
- `genre : 'action'.`

## Renderize um formulário dentro de `AddMovie`

Dentro desse formulário haverá campos usados para preencher informações do novo cartão a ser adicionado na biblioteca.

### Renderize um input do tipo texto dentro do formulário em `AddMovie` para obter o título do novo filme

- O input deve ter uma label associada com o texto: **"Título"**;
- O input deve ter seu valor inicial provido pelo estado inicial do componente, via `title` ;
- A propriedade `onChange` deve atualizar o estado de `AddMovie` , atribuindo a `title` o atual título contido no input.

### Renderize um input do tipo texto dentro do formulário em `AddMovie` para obter o subtítulo do novo filme

- O input deve ter uma label associada com o texto: **"Subtítulo"**;
- O input deve ter seu valor inicial provido pelo estado inicial do componente, via `subtitle` ;
- A propriedade `onChange` deve atualizar o estado de `AddMovie` , atribuindo a `subtitle` o atual subtítulo contido no input.

### Renderize um input do tipo texto dentro do formulário em `AddMovie` para obter o caminho da imagem do novo filme

- O input deve ter uma label associada com o texto: **"Imagem"**;
- O input deve ter seu valor inicial provido pelo estado inicial do componente, via `imagePath` ;
- A propriedade `onChange` deve atualizar o estado de `AddMovie` , atribuindo a `imagePath` o atual caminho da imagem contido no input.

Renderize uma `textarea` dentro do formulário em `AddMovie` para obter a sinopse do novo filme

- A `textarea` deve ter uma label associada com o texto: **"Sinopse"**;
- A `textarea` deve ter seu valor inicial provido pelo estado inicial do componente, via `storyline` ;
- A propriedade `onChange` deve atualizar o estado de `AddMovie` , atribuindo a `storyline` a sinopse atual contida na `textarea` .

Renderize um `input` do tipo `number` dentro do formulário em `AddMovie` para obter a avaliação do novo filme

- O `input` deve ter uma label associada com o texto: **"Avaliação"**;
- O `input` deve ter seu valor inicial provido pelo estado inicial do componente, via `rating` ;
- A propriedade `onChange` deve atualizar o estado de `AddMovie` , atribuindo a `rating` a avaliação atual contida no input.

Renderize um `select` do formulário em `AddMovie` para selecionar o gênero do novo filme

- O `select` deve ter uma label associada com o texto: **"Gênero"**;
- O `select` deve ter seu valor inicial provido pelo estado inicial do componente, via `genre` ;
- A propriedade `onChange` deve atualizar o estado de `AddMovie` , atribuindo a `genre` o gênero atual selecionado;
- O `select` deve renderizar três tags `option` , com as opções de filtragem por gênero, na seguinte ordem:
  - `Ação` , com o valor `action` ;
  - `Comédia` , com o valor `comedy` ;
  - `Suspense` , com o valor `thriller` .

Renderize um botão do formulário em `AddMovie` para fazer uso dos dados do novo filme, contidos no estado de `AddMovie`

- O botão precisa ter escrito o seguinte texto: **"Adicionar filme"**;

- A propriedade `onClick` do botão invoca uma função definida por você, em `AddMovie`, que:
  - Executa a callback passada para o componente `AddMovie` via props, chamada `onClick`, que recebe como parâmetro o estado atual de `AddMovie`;
  - Reseta o estado de `AddMovie`, voltando para o inicial, conforme mencionado anteriormente.

## Crie um componente chamado `MovieLibrary`

Esse componente renderizará a biblioteca de filmes, com a possibilidade de filtrar por filmes e adicionar um filme à biblioteca.

`MovieLibrary` deve receber como props:

- `movies`, um array

## Configure o estado inicial do componente `MovieLibrary`

O componente `MovieLibrary` possui como estado as seguintes propriedades:

- `searchText`: guarda o texto de busca por filmes;
- `bookmarkedOnly`: um *boolean* que guarda se é para filtrar por filmes favoritos ou não;
- `selectedGenre`: guarda o gênero do filme selecionado para poder fazer a filtragem;
- `movies`: guarda a lista de filmes.

Ou seja, o estado de `MovieLibrary` contém a lista de filmes e os filtros a serem aplicados sobre a listagem.

O estado inicial do componente `MovieLibrary` deve ser:

- `searchText`: "";
- `bookmarkedOnly`: false;
- `selectedGenre`: "";
- `movies`: a lista de filmes passadas pela props `movies`.

## Renderize `SearchBar` dentro de `MovieLibrary`

- `searchText` oriundo do estado de `MovieLibrary` deve ser passado para a prop `searchText` de `SearchBar`;

- A callback para atualizar o estado de `MovieLibrary` em `searchText` precisa ser passada para `SearchBar` ;
- `bookmarkedOnly` oriundo do estado de `MovieLibrary` deve ser passado para a prop `bookmarkedOnly` de `SearchBar` ;
- A callback para atualizar o estado de `MovieLibrary` em `bookmarkedOnly` precisa ser passada para `SearchBar` ;
- `selectedGenre` oriundo do estado de `MovieLibrary` deve ser passado para a prop `selectedGenre` de `SearchBar` ;
- A callback para atualizar o estado de `MovieLibrary` em `selectedGenre` precisa ser passada para `SearchBar` .

### Renderize `MovieList` dentro de `MovieLibrary`

- Deve passar para a prop `movies` de `MovieList` todos os filmes filtrados;
- Quando o estado para `bookmarkedOnly` é falso, não é alterada a listagem de filmes a ser renderizada;
- Quando o estado para `bookmarkedOnly` é verdadeiro, deve ser renderizado por `MovieList` somente filmes favoritados;
- Quando o estado para `selectedGenre` é vazio, não é alterada a listagem de filmes a ser renderizada;
- Quando o estado para `selectedGenre` não é vazio, deve ser renderizado somente filmes com o mesmo gênero;
- Quando o estado para `searchText` é vazio, não é alterada a listagem de filmes a ser renderizada;
- Quando o estado para `searchText` não é vazio, deve ser renderizado por `MovieList` filmes que satisfaçam a uma das condições abaixo:
  - Filmes cujo título contém o que está presente em `searchText` , **ou**;
  - Filmes cujo subtítulo contém o que está presente em `searchText` , **ou**;
  - Filmes cuja sinopse contém o que está presente em `searchText` .

### Renderize `AddMovie` dentro de `MovieLibrary`

- A callback que permite adicionar um novo filme ao final da lista precisa ser passada para `AddMovie` .

# Instruções para entregar seu projeto:

## ANTES DE COMEÇAR A DESENVOLVER:

### 1. Clone o repositório

- `git clone git@github.com:tryber/sd-03-block13-project-movie-card-library-stateful.git`.
- Entre na pasta do repositório que você acabou de clonar:
  - `cd sd-03-block13-project-movie-card-library-stateful`

### 2. Instale as dependências, inicialize o projeto e rode os testes

- Instale as dependências:
  - `npm install`
- Inicialize o projeto:
  - `npm start` (uma nova página deve abrir no seu navegador com um texto simples)
- Verifique que os testes estão executando:
  - `npm test` (os testes devem rodar e falhar)

### 3. Crie uma branch a partir da branch `master`

- Verifique que você está na branch `master`
  - Exemplo: `git branch`
- Se não estiver, mude para a branch `master`
  - Exemplo: `git checkout master`
- Agora, crie uma branch onde você vai guardar os `commits` do seu projeto
  - Você deve criar uma branch no seguinte formato: `nome-de-usuario-nome-do-projeto`
  - Exemplo: `git checkout -b joaozinho-movie-card-library-stateful`

### 4. Faça alterações em algum dos componentes que precisam de implementação, por exemplo o `MovieLibrary` em `src/components`:

```
import React, { Component } from 'react';

import MovieList from './MovieList';
import SearchBar from './SearchBar';
import AddMovie from './AddMovie';
```

```

class MovieLibrary extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <h2> My awesome movie library </h2>
        <SearchBar />
        <MovieList movies={this.props.movies} />
        <AddMovie />
      </div>
    );
  }
}

export default MovieLibrary;

```

5. Adicione as mudanças ao *stage* do Git e faça um `commit`

- Verifique que as mudanças ainda não estão no *stage*
  - Exemplo: `git status` (deve aparecer listado o arquivo `src/components/MovieLibrary.jsx` em vermelho)
- Adicione o arquivo alterado ao *stage* do Git
  - Exemplo:
    - `git add .` (adicionando todas as mudanças - *que estavam em vermelho* - ao *stage* do Git)
    - `git status` (deve aparecer listado o arquivo `src/components/MovieLibrary.jsx` em verde)
- Faça o `commit` inicial
  - Exemplo:
    - `git commit -m 'iniciando o projeto. VAMOS COM TUDO :rocket:'` (fazendo o primeiro `commit`)
    - `git status` (deve aparecer uma mensagem tipo *nothing to commit*)

6. Adicione a sua branch com o novo `commit` ao repositório remoto



- Usando o exemplo anterior: `git push -u origin joaozinho-movie-cards-library-stateful`

7. Crie um novo `Pull Request` (PR)

- Vá até a página de *Pull Requests* do [repositório no GitHub](#)
- Clique no botão verde "New pull request"

- Clique na caixa de seleção "*Compare*" e escolha a sua branch **com atenção**
  - Clique no botão verde "*Create pull request*"
  - Adicione uma descrição para o *Pull Request* e clique no botão verde "*Create pull request*"
  - **Não se preocupe em preencher mais nada por enquanto!**
  - Volte até a [página de Pull Requests do repositório](#) e confira que o seu *Pull Request* está criado
- 

## DURANTE O DESENVOLVIMENTO

-  **PULL REQUESTS COM ISSUES NO CODE CLIMATE NÃO SERÃO AVALIADAS, ATENTE-SE PARA RESOLVÊ-LAS ANTES DE FINALIZAR O DESENVOLVIMENTO!** 
  - Faça `commits` das alterações que você fizer no código regularmente
  - Lembre-se de sempre após um (ou alguns) `commits` atualizar o repositório remoto
  - Os comandos que você utilizará com mais frequência são:
    - i. `git status` (para verificar o que está em vermelho - fora do stage - e o que está em verde - no stage)
    - ii. `git add` (para adicionar arquivos ao stage do Git)
    - iii. `git commit` (para criar um commit com os arquivos que estão no stage do Git)
    - iv. `git push -u nome-da-branch` (para enviar o commit para o repositório remoto na primeira vez que fizer o `push` de uma nova branch)
    - v. `git push` (para enviar o commit para o repositório remoto após o passo anterior)
- 

## DEPOIS DE TERMINAR O DESENVOLVIMENTO

Para "**entregar**" seu projeto, siga os passos a seguir:

- Vá até a página **DO SEU Pull Request**, adicione a label de "*code-review*" e marque seus colegas
  - No menu à direita, clique no *link* "**Labels**" e escolha a *label* **code-review**
  - No menu à direita, clique no *link* "**Assignees**" e escolha **o seu usuário**
  - No menu à direita, clique no *link* "**Reviewers**" e digite `students`, selecione o time `tryber/students-sd-03`



Se ainda houver alguma dúvida sobre como entregar seu projeto, [aqui tem um video explicativo](#).

⚠️ Lembre-se que garantir que todas as *issues* comentadas pelo CodeClimate estão resolvidas! ⚠️

---

## REVISANDO UM PULL REQUEST



À medida que você e as outras pessoas que estudam na Trybe forem entregando os projetos, vocês receberão um alerta via Slack para também fazer a revisão dos Pull Requests dos seus colegas. Fiquem atentos às mensagens do "Pull Reminders" no Slack!

Use o material que você já viu sobre [Code Review](#) para te ajudar a revisar os projetos que chegaram para você.

### Releases

No releases published

---

### Packages

No packages published

---

### Languages

