

ReviewHub: Sistema de Avaliação de Filmes

1. Visão Geral e Tecnologias

Sobre o Projeto O ReviewHub é uma aplicação web focada em avaliações de filmes. O objetivo é fornecer uma plataforma simples para realizar operações de **CRUD** (Create, Read, Update, Delete) em reviews de filmes.

Stack Tecnológico

- **Linguagem:** Python
- **Framework Web:** Flask
- **Banco de Dados:** SQLite
- **Gerenciador de Dependências:** Poetry
- **Frontend:** HTML5, CSS3, JavaScript

2. Como inicializar o projeto

Clonando o Repositório

Antes de tudo, abra o seu terminal na pasta onde deseja salvar o projeto e execute **um** dos comandos abaixo, dependendo da sua preferência:

Opção 1: Via HTTPS (Padrão)

```
git clone https://github.com/zuckii/reviewhub.git
```

Opção 2: Via SSH (Requer chaves SSH configuradas)

```
git clone git@github.com:zuckii/reviewhub.git
```

Opção 3: Via GitHub CLI

```
gh repo clone zuckii/reviewhub
```

Com isso, entre na pasta **reviewhub**:



```
# No terminal
2
3 cd reviewhub
```

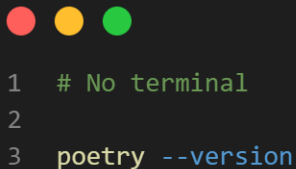
Pré-requisitos

Antes de iniciar a aplicação, certifique-se de que seu ambiente possui os seguintes requisitos instalados:

- **Python:** Versão 3.12.3 ou superior (conforme especificado no [pyproject.toml](#)).
- **Poetry:** Versão 2.2.1 ou superior (conforme gerado no [poetry.lock](#)).

Passo a passo de inicialização

Verificação da Instalação do Poetry Primeiramente, verifique se o gerenciador de dependências está instalado corretamente:

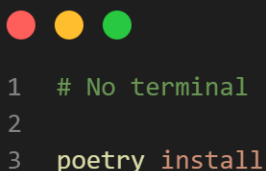


```
1 # No terminal
2
3 poetry --version
```

Caso não esteja instalado, siga a documentação oficial:

python-poetry.org/docs/#installation

Instalação das Dependências Navegue até a raiz do projeto **ReviewHub** e execute:

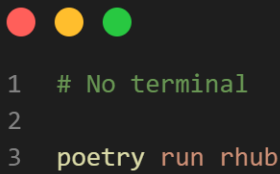


```
1 # No terminal
2
3 poetry install
```

Este comando realizará as seguintes ações:

- Criará um ambiente virtual isolado (Python $\geq 3.12.3$).
- Instalará todas as bibliotecas listadas no [pyproject.toml](#).
- Configurarão o ambiente de desenvolvimento.

Execução da Aplicação O ReviewHub possui um script de entrada personalizado. Para iniciar, utilize:

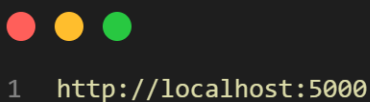


```
1 # No terminal
2
3 poetry run rhub
```

O que acontece ao executar este comando?

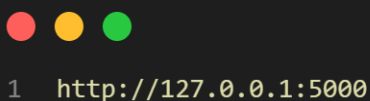
- O script *rhuh* executa a função *run()* do módulo *app*.
- O servidor Flask é iniciado com as dependências carregadas.
- O banco de dados SQLite é inicializado (caso seja a primeira execução).

Acesso à Aplicação Após a mensagem de sucesso no terminal, acesse a aplicação pelo navegador:



```
1 http://localhost:5000
```

ou




```
1 http://127.0.0.1:5000
```

Estrutura de Dependências Instaladas

Ao rodar o *poetry install*, as seguintes bibliotecas principais são configuradas no ambiente virtual:

- **Flask (3.1.2)**: Framework web principal.
- **Werkzeug (3.1.3)**: Biblioteca WSGI utilitária.
- **Jinja2 (3.1.6)**: Motor de templates para renderização HTML.
- **MarkupSafe (3.0.3)**: Segurança contra injeção de código em templates.
- **Click (8.3.0)**: Framework para interfaces de linha de comando.
- **ItsDangerous (2.2.0)**: Assinatura segura de dados.
- **Blinker (1.9.0)**: Sistema de sinalização (Signals).
- **Colorama (0.4.6)**: Suporte a cores no terminal (Windows).

Para verificar a lista completa e as versões instaladas no seu ambiente, execute:




```
1 # No terminal
2
3 poetry show
```

Solução de Problemas Comuns

Se encontrar dificuldades ao rodar o comando principal, tente as soluções abaixo:


Problema: O comando *poetry run rhub* falha.

- *Alternativa 1 (Flask direto):*



```
1 # No terminal
2
3 poetry run flask run
```


- *Alternativa 2 (Python direto):*



```
1 # No terminal
2
3 poetry run python app.py
```

Problema: Erro de versão do Python.

- Verifique se sua versão é compatível ($\geq 3.12.3$):



```
1 # No terminal
2
3 python --version
```


Problema: Erros de dependências ou módulos não encontrados.

- Tente atualizar o ambiente:



```
1 # No terminal
2
3 poetry update
```

- Se persistir, recrie o ambiente virtual do zero:



```
1 # No terminal
2
3 poetry env remove python
4 poetry install
```

Problema: Módulo *app* não encontrado.

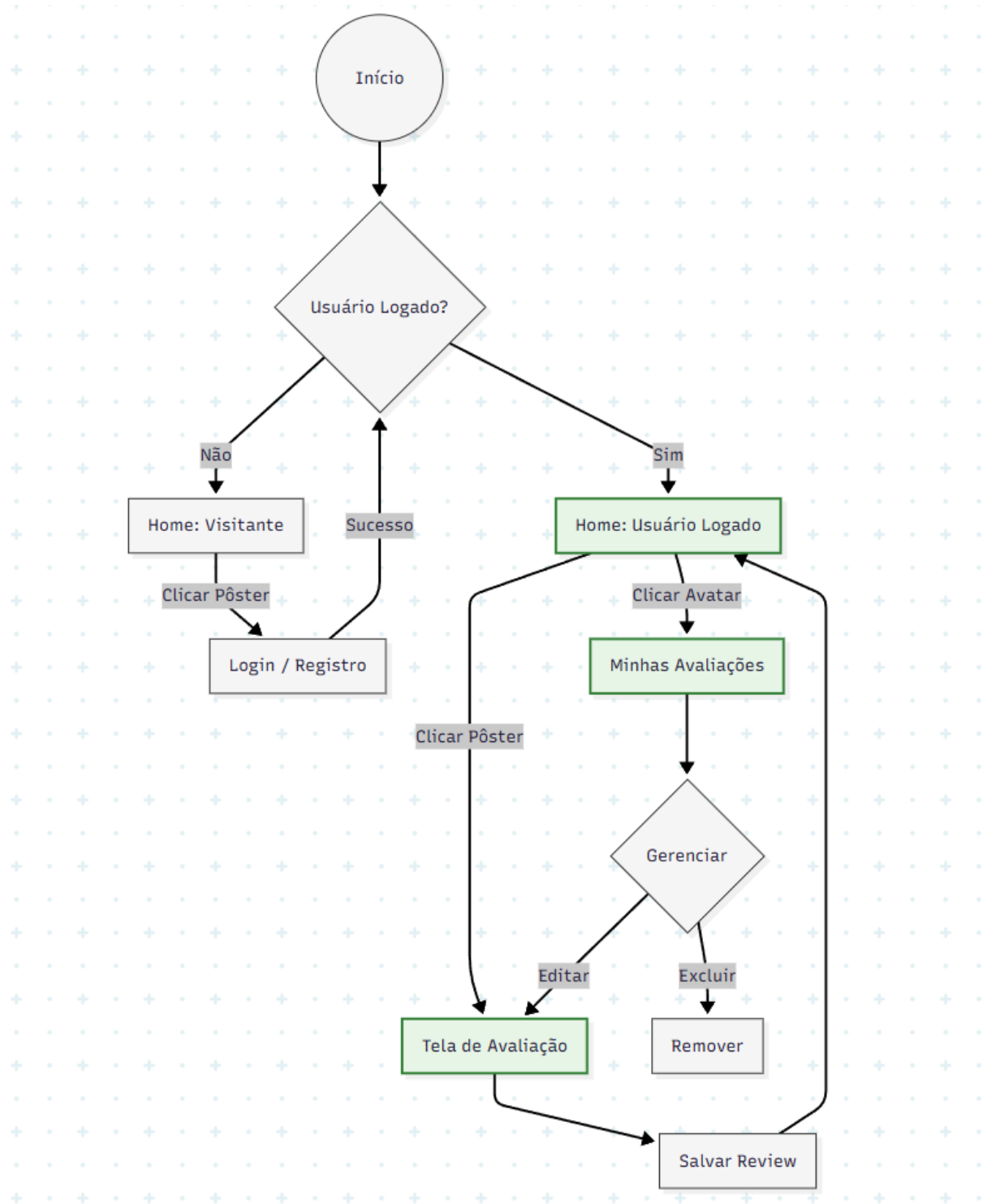
- Verifique se o arquivo *app.py* existe na raiz do projeto e se contém a função *run()*.

3. Caminho do Usuário (User Path)

Este fluxo descreve a navegação completa de um usuário interagindo com o sistema:

- a. **Home (Read - Listagem Pública):** O usuário acessa a página inicial e visualiza a galeria de pôsteres de todos os filmes cadastrados. Nesta tela, é possível filtrar os filmes por gênero (Ação, Drama, Comédia, etc.) para facilitar a busca.
- b. **Verificação de Identidade (Autenticação):** Ao clicar no pôster de um filme para avaliá-lo, o sistema verifica se o usuário está logado:
 - **Usuário Logado:** É direcionado imediatamente para a tela de avaliação.
 - **Usuário Visitante:** É redirecionado para a página de **Login/Registro**. Após se autenticar, o sistema o devolve para a página inicial.
- c. **Adicionar Avaliação (Create):** Já autenticado e na tela do filme escolhido, o usuário preenche sua review com nota e uma Descrição sobre o filme, salvando sua opinião no sistema.
- d. **Perfil do Usuário (Read - Pessoal):** Ao clicar no ícone do seu perfil/avatar no menu superior, o usuário acessa a área "Minhas Avaliações". Aqui, ele visualiza uma lista exclusiva contendo apenas as críticas feitas por ele.
- e. **Gestão de Reviews (Update & Delete):** Dentro da área de perfil, cada avaliação listada possui opções de gerenciamento:
 - **Editar:** Permite alterar a nota ou o texto da crítica caso o usuário mude de opinião.
 - **Excluir:** Permite remover a avaliação permanentemente do sistema.

Diagrama de uso do *reviewHub*



4. Explicando o Banco de Dados

a. Estrutura das Tabelas:

i. Usuário:

1. Armazena os dados de quem acessa o sistema.
2. O campo *username* possui uma restrição **UNIQUE**, impedindo que dois usuários tenham o mesmo login.
3. A senha é armazenada como hash.

ii. Gênero:

1. Serve para evitar repetição. O nome do gênero (ex: "Ação") fica guardado apenas uma vez na tabela Gênero. A tabela Filmes usa apenas o número (ID) para apontar para ele, deixando o banco mais leve e organizado.

iii. Filmes:

1. Contém os dados dos filmes (nome e imagem).
2. Possui uma Chave Estrangeira (FK) *genero_id* que aponta para a tabela Gênero, criando uma relação direta de dependência.

iv. Avaliação:

1. Ela conecta Usuários e Filmes.
2. É formada por uma Chave Primária Composta (*usuario_id*, *filme_id*), o que garante que um usuário só pode avaliar o mesmo filme uma única vez.
3. Possui uma restrição de integridade (CHECK) que obriga a nota a estar entre 0 e 5.

b. Relacionamentos

i. Gênero e Filmes (1:N):

1. *Lógica*: Um Gênero pode ter vários Filmes, mas cada Filme pertence a apenas um Gênero principal neste modelo.

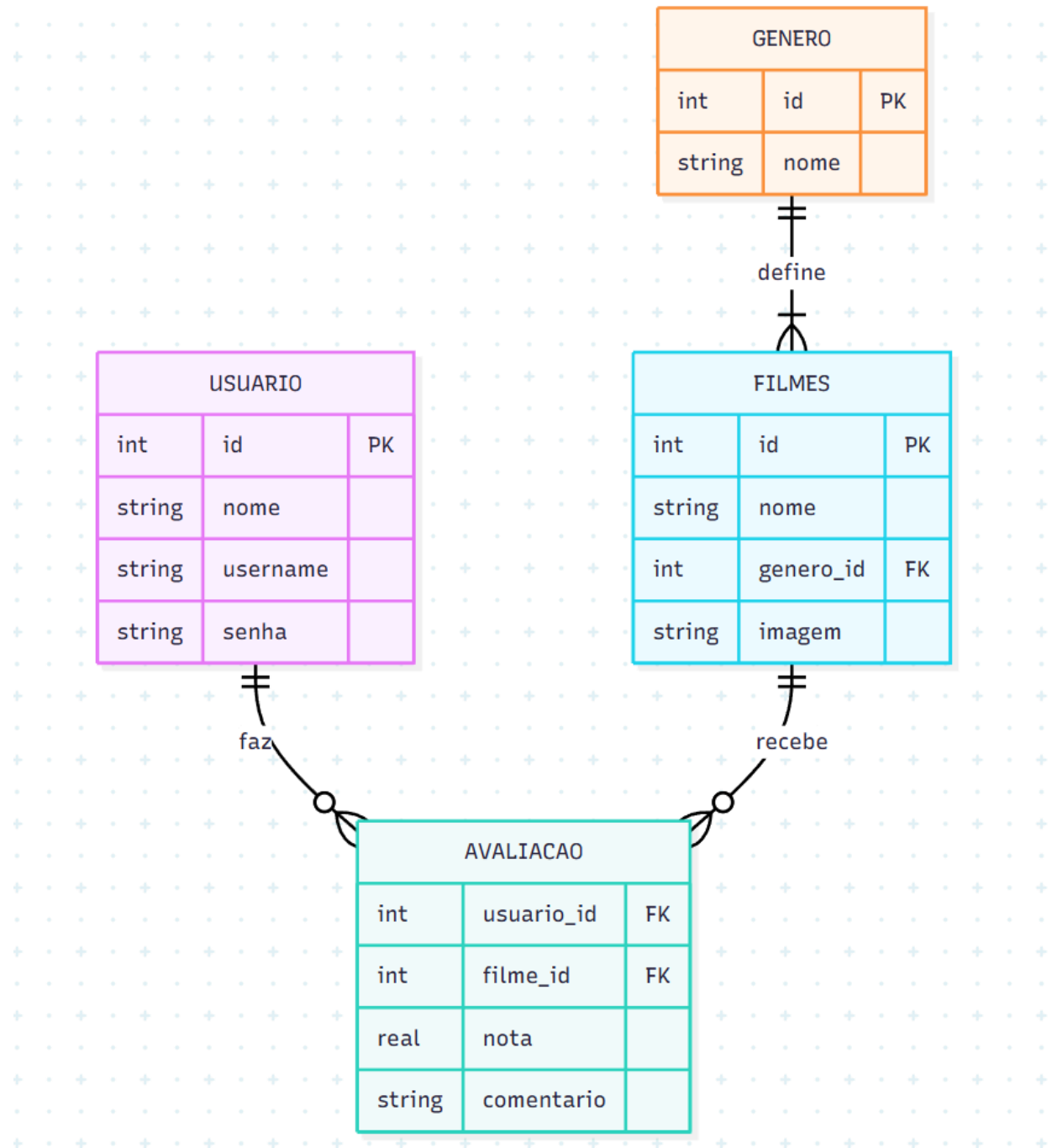
ii. Usuários e Filmes (N:N):

1. *Lógica*: Um Usuário pode avaliar vários Filmes, e um Filme pode receber avaliações de vários Usuários.
2. *Implementação*: Essa relação "N:N" é resolvida fisicamente pela tabela intermediária Avaliação.

c. Regras de Negócio Implementadas no Banco

- i. **Unicidade**: Nomes de filmes e gêneros não podem se repetir **UNIQUE**.
- ii. **Validação de Dados**: O banco rejeita automaticamente qualquer nota negativa ou maior que 5.
- iii. **Integridade Referencial**: Se tentar apagar um usuário que fez avaliações, o banco pode impedir ou exigir tratamento (dependendo da configuração de **ON DELETE** do SQLite, mas as FKs estão ativas).

Visualização da estrutura do Banco de Dados usado no nosso projeto:



5. Tutorial CRUD do reviewHub

a. CREATE (Criar Avaliação de Filme):

- i. A criação de uma avaliação ocorre na página de detalhes do filme. O sistema impede avaliações duplicadas ou inválidas através de verificações de sessão e validação de input.
- ii. A rota captura os dados e o ID do usuário da sessão.
- iii. Verifica se a nota está entre 0 e 5.
- iv. Chama `insert_review` para persistir no banco.

Exemplo de código pego de [reviewhub/routes/movies.py](#).

```
1  @movies_bp.post("<int:id>")
2  def details(id):
3      user_id = session["user_id"]
4
5      # Captura os dados enviados pelos campos do formulário HTML (request.form).
6      # 'rating' é o valor das estrelas e 'review' é o texto do comentário.
7      nota = float(request.form.get("rating"))
8      comentario = request.form.get("review")
9
10     # Valida se a nota está dentro do intervalo permitido (0 a 5).
11     if nota < 0 or nota > 5:
12         flash("Nota inválida. Deve ser entre 0 e 5.", "nok")
13         return redirect(url_for("movies.details_page", id=id))
14
15     # Bloco try/except para tentar realizar a operação no banco de dados.
16     try:
17         # Tenta inserir a avaliação no banco de dados.
18         insert_review(user_id, id, nota, comentario)
19
20         # Se funcionar, exibe uma mensagem de sucesso ("ok").
21         flash("Avaliação enviada com sucesso!", "ok")
22
23     except Exception as e:
24         # Se der erro, captura a exceção e exibe uma mensagem de erro ("nok").
25         flash(f"Erro ao salvar avaliação: {e}", "nok")
26         return redirect(url_for("movies.details_page", id=id))
27
28     # Se tudo der certo, redireciona o usuário para a página inicial (homepag
29     e).
30     return redirect(url_for("home.homepage"))
```

b. READ(Ler Filmes e Avaliações):

- i. A leitura é dividida em listar filmes (página inicial) e listar avaliações específicas (página de detalhes ou "Minhas Avaliações").
- ii. A rota captura os dados e o ID do usuário da sessão.
- iii. Busca os dados do filme (`get_all_movies`) e as avaliações associadas (`get_rating_by_movie`).
- iv. Calcula a média das notas para exibição.
- v. Renderiza o template injetando os objetos.

Exemplo de código pego de [reviewhub/routes/movies.py](#)

```
1 @movies_bp.route("/mine")
2 def mine_reviews_page():
3     # Rota responsável por exibir apenas as avaliações do usuário logado.
4
5     # Verifica se existe um usuário logado na sessão.
6     # Caso não exista, impede o acesso e redireciona para a página de login.
7     if not session.get('user_id'):
8         flash("Para listar suas avaliações é necessário realizar login.", "nok")
9         return redirect(url_for('auth.login_page'))
10
11     # Obtém do banco todas as avaliações vinculadas ao ID do usuário logado.
12     reviews = get_reviews_by_user(session.get('user_id'))
13
14     # Renderiza o HTML enviando a lista de reviews para ser exibida na tela.
15     return render_template("movies/mine.html", reviews=reviews)
```

c. UPDATE(Atualizar Avaliação):

- i. Permite que o usuário edite sua nota e comentário. O backend utiliza uma rota específica para processar a atualização.
- ii. O formulário envia os dados para a rota `/update/<id>`.
- iii. O backend valida a sessão do usuário.
- iv. Executa a função `update_review` no banco de dados, que altera o registro onde o `usuario_id` e `filme_id` coincidem.

Exemplo de código pego de [reviewhub/routes/movies.py](#)

```
1 @movies_bp.post("/update/<int:id>")
2 def update_review_route(id):
3     # Verificação de Autenticação
4     user_id = session.get("user_id")
5
6     if not user_id:
7         flash("Para atualizar avaliações é necessário realizar login.", "nok")
8         return redirect(url_for('auth.login_page'))
9
10    # Coleta de Dados do Formulário
11    nota = float(request.form.get("rating"))
12    comentario = request.form.get("review")
13
14    # Validação de Dados
15    if nota < 0 or nota > 5:
16        flash("Nota inválida. Deve ser entre 0 e 5.", "nok")
17        return redirect(url_for("movies.details_page", id=id))
18
19    # Atualização da Avaliação no Banco de Dados
20    try:
21        # Chama a função controller/service que executa a query SQL
22        update_review(user_id, id, nota, comentario)
23        flash("Avaliação atualizada com sucesso!", "ok")
24    except Exception as e:
25        # Captura erros de banco de dados ou lógica e avisa o usuário
26        flash(f"Erro ao atualizar avaliação: {e}", "nok")
27        return redirect(url_for("movies.details_page", id=id))
28
29    return redirect(url_for("home.homepage"))
```

d. DELETE(Remover Avaliação):

- i. A remoção é acionada via JavaScript (Fetch API) no frontend, permitindo uma experiência mais fluida sem recarregar formulários tradicionais.
- ii. O usuário clica no botão de remover em [mine.html](#).
- iii. O JavaScript envia uma requisição com método **DELETE**.
- iv. A rota Flask intercepta o método DELETE, verifica se a avaliação existe e a remove.

Exemplo de código pego de [reviewhub/routes/movies.py](#)

```
1 @movies_bp.delete("/<int:id>")
2 def remove_review(id):
3
4     # Impede que usuários não logados tentem deletar dados
5     if not session.get('user_id'):
6         flash("Para remover avaliações é necessário realizar login.", "nok")
7         return redirect(url_for('auth.login_page'))
8
9     # Busca TODAS as reviews do usuário para confirmar se a review deste filme ex
10    iste
11    reviews = get_reviews_by_user(session.get('user_id'))
12
13    # Itera sobre a lista para encontrar a review correspondente ao ID do filme
14    review = next((r for r in reviews if r['filme_id'] == id), None)
15
16    # Se a review existir, deleta-a
17    if review:
18        delete_review(session.get('user_id'), id)
19        flash("Avaliação removida com sucesso!", "ok")
20    else:
21        # Caso o usuário tente deletar uma review que não existe ou não é dele
22        flash("Avaliação não encontrada.", "nok")
23
24    # Retorna para a página "Minhas Avaliações"
25    return redirect(url_for("movies.mine_reviews_page"))
```

Segue link do vídeo de demonstração da aplicação.

https://youtu.be/eDT7_1u0ZX0