

# Sistema de Recomendação de Filmes

Adriana Lopes, Diana Carrilho, Henrique Faria, Paulo Barbosa

Universidade do Minho, Mestrado Integrado em Engenharia Informática e Mestrado em Matemática e Computação

**Resumo** Neste trabalho abordou-se o tema "Sistemas de Recomendação". Este projeto insidiu sobre um dataSet de filmes encontrado no Kaggle e guardado em ficheiros excel. Para a realização deste projeto usaram-se várias linguagens de programação e tecnologias, nomeadamente Flask(Python), HTML, CSS, Jinja2, JavaScript e MongoDB. O Mongo foi usado para guardar o nosso dataSet de forma a que este no futuro posso escalar horizontalmente e também possibilitar um acesso mais rápido aos dados. O HTML, o CSS, o Jinja2 e o JavaScript foram usados para fazer as páginas web e programar o seu comportamento. O Flask foi usado para fazer a ligação entre FrontEnd e Backend e para controlar as mudanças de estado das páginas web.

**Palavras-chave:** Sistemas de Recomendação · Flask · Python · HTML · CSS · Jinja2 · JavaScript · MongoDB · Filmes

## Introdução

Neste trabalho foi explorada a aplicação dos Sistemas de Recomendação a uma base de dados de filmes. O primeiro passo para este objetivo passou por planear os requisitos dos utilizadores da nossa base de dados que usem o nosso sistema de recomendação. Este requisitos são: obter uma recomendação baseada no que os top X utilizadores semelhantes ao alvo viram e obter uma recomendação baseada no conteúdo que o alvo viu e gostou (por exemplo por gênero de filme). Para os casos de cold-start de utilizadores foram adicionados métodos para apresentação dos top 10 filmes mais bem cotados pela IMDB e os 15 filmes mais bem cotados por utilizadores do sistema e para os casos de cold-start dos filmes foi criado um método para devolver os 15 filmes mais recente ainda sem visualizações por data de lançamento(do mais recente para o mais antigo). Para além disso foi introduzido uma search bar que permite ao utilizador da aplicação utilizar um sistema de recomendação baseado em restrições, iterativo. Adicionalmente juntaram-se os requisitos de consultar filmes por ano de lançamento ou por gênero.

Para permitir uma experiência personalizada aos utilizadores foi necessário identifica-los inequivocamente e para isso tivemos de implementar um sistema de autenticação no qual o utilizador é identificado por um email único e ao qual é associado um id usado para guardar as suas avaliações dos filmes vistos.

## 1 Descrição do dataset escolhido

O *dataset* escolhido é composto por dois ficheiros, "movies\_\_metadata" e "small\_\_ratings".

O "movies\_\_dataset", guarda informações sobre cada filme usando os seguintes atributos:

- **adult:** Carateriza se um filme é para maiores de 18 anos;
- **homepage:** Link para a página oficial do filme;
- **movieId:** Identificação do filme na base de dados;
- **imdb\_id:** Identificação do filme no IMDB;
- **original\_language:** Língua do filme;
- **original\_title:** Título do filme na língua original;
- **overview:** Resumo do filme;
- **popularity:** Popularidade do filme;
- **poster\_path:** Imagem do filme;
- **release\_date:** Data da lançamento do filme;
- **runtime:** Duração do filme;
- **status:** Estado de lançamento do filme;
- **tagline:** Frase icónica do filme;
- **title:** Título do filme traduzido para inglês;
- **vote\_average:** Cotação média do filme;
- **vote\_count:** Número de votações a que o filme foi sujeito;
- **Animation, ..., Foreign:** Géneros dos filmes, o filme pertence a um género se estives identificado com 1 nessa coluna, 0 caso contrário (colunas 16 à 35);
- **ColectionId:** Identificação da coleção;
- **Colection:** Coleção à qual um filme pertence;
- **English, Deutsh, Français, Espanhol:** Caso o filme tenha sido traduzido para uma lingua este é identificado com 1 na respetiva coluna, 0 caso contrário.

Este dataSet dará origem á coleção films na base de dados Mongo.

O "small\_\_ratings" guarda as avaliações dos utilizadores face aos filmes vistos e é composto pelos seguintes atributos:

- **userID:** Identificação do utilizador;
- **movieId:** Identificação do filme;
- **rating:** Cotação que o utilizador deu ao filme;
- **timestamp:** Data da cotação.

Este dataSet dará origem á coleção userRating na base de dados Mongo.

Para além destes datasets foi criado um chamado users que contem as informações da conta de cada utilizador.

- **id:** Identificação do utilizador;
- **username:** Nome do utilizador;
- **email:** Email do utilizador;
- **password:** Password do utilizador;

Este dataSet dará origem á coleção users na base de dados Mongo.

## 2 Sistemas de Recomendação implementados

### 2.1 Sistemas baseados em filtros colaborativos

Nestes sistemas foram utilizadas as várias avaliações dos utilizadores para se fazerem recomendações.

#### Filtros colaborativos baseados em utilizadores

Neste caso, as classificações fornecidas por utilizadores com os mesmos gostos de A são usadas para fazer recomendações para A. O método que realiza este filtro chama-se *colab\_filtering()*, de seguida apresenta-se o método e explica-se o seu funcionamento.

```
def colab_filtering():
    user = mongo.db.users.find_one({"email": user_global_id})
    #Cold Start Users
    if mongo.db.userRating.find({"userId": user["id"]}).count() == 0:
        return top10_avg()
    moviesToCompare = mongo.db.userRating.find({"userId": user["id"]})
    similar = similar_film = filmes = []
    res = []
    res2 = []
    row = ten = 0
    passa = 0
    #encontra pessoas que viram os mesmos filmes
    for mv in moviesToCompare:
        similar += mongo.db.userRating.find({"movieId": mv["movieId"]})
    for s in similar:
        s["dif"] = abs(mv["rating"] - s["rating"])
    df = pd.DataFrame(similar)
    new = df.groupby(['userId'], as_index=False)["dif"].mean().to_dict('r')
    similar2 = sorted(new, key=lambda x: x['dif'])
    #lista dos filmes vistos pelos 10 utilizadores mais parecidos
    for s in similar2:
        user_id = s['userId']
        if row == 0:
            row = s["userId"]
        elif ten > 10:
            break
        elif user_id != row:
            filmes = mongo.db.userRating.find({"userId": row, "movieId": {"$lt": 1000}, "rating": {"$gt": 2.5}}, {"movieId": 1, "_id": 0}).limit(15)
            for f in filmes:
                film = mongo.db.films.find({"movieId": f["movieId"]})
                if film != None:
                    res += film
                    ten += 1
            for r in res:
                passa = 0
                for m in moviesToCompare:
                    if r["movieId"] == m["movieId"]:
                        passa = 1
                        break
                if passa == 0:
                    res2 += r
                else:
                    ten -= 1
            row = s["userId"]
            ten += 1
        else:
            row = row
    return render_template("personalHome.html", title="personalHome", films = res[:15])
```

Figura 1. Função *colaborative\_filtering()*

Neste método, a primeira coisa a fazer é recolher os documentos com as avaliações do utilizador aos filmes por ele vistos. Nesta fase o utilizador pode não ter visto filmes ainda e solicitar o filtro colaborativo, neste caso esta função retorna-lhe os filmes mais bem cotados pelo IMDB (que têm mais chances de ser do interesse do utilizador).

Caso o utilizador já tenha visto um ou mais filmes estes são recolhidos e são procurados na base de dados documentos de outros utilizadores que tenham visto os mesmos filmes, sendo estes guardados num array. De seguida este array é percorrido e é calculada a diferença entre as avaliações de cada utilizador e o nosso alvo guardando-as numa coluna extra. Após este processo é calculada a média das diferenças por utilizador e o array resultante com duas posições (id utilizador, diferença) é ordenado por ordem crescente de diferença. Por fim basta procurar quais os filmes avaliados por cada utilizador, percorrendo o array de utilizadores semelhantes, dos quais estes gostaram tendo o cuidado de não inserir filmes repetidos ou que o alvo já tenha visto.

## 2.2 Sistemas baseados em conteúdo

Nestes sistemas, as avaliações do utilizador, assim como os géneros dos filmes de que ele gosta, são combinados para fazer as recomendações. O método para realizar esta abordagem chama-se *filmWatchBased()*, de seguida apresenta-se o método e explica-se o seu funcionamento.

```

def filmWatchBased():
    user = mongo.db.users.find_one({"email":user_global_id})
    moviesSeen = mongo.db.userRating.find({"userId": user["id"], "rating": {"$gt":2}},{"movieId":1,"_id":0})
    genresAgregation = []
    genres = []
    films = []
    rec = []
    res = []
    for m in moviesSeen:
        genresAgregation += mongo.db.films.find({"movieId": m["movieId"]})
    for g in genresAgregation:
        if g["Animation"] == 1 and "Animation" not in genres:
            genres.append("Animation")
        if g["Adventure"] == 1 and "Adventure" not in genres:
            genres.append("Adventure")
        if g["Romance"] == 1 and "Romance" not in genres:
            genres.append("Romance")
        if g["Comedy"] == 1 and "Comedy" not in genres:
            genres.append("Comedy")
        if g["Action"] == 1 and "Action" not in genres:
            genres.append("Action")
        if g["Family"] == 1 and "Family" not in genres:
            genres.append("Family")
        if g["History"] == 1 and "History" not in genres:
            genres.append("History")
        if g["Drama"] == 1 and "Drama" not in genres:
            genres.append("Drama")
        if g["Crime"] == 1 and "Crime" not in genres:
            genres.append("Crime")
        if g["Fantasy"] == 1 and "Fantasy" not in genres:
            genres.append("Fantasy")
        if g["Science Fiction"] == 1 and "Science Fiction" not in genres:
            genres.append("Science Fiction")
        if g["Thriller"] == 1 and "Thriller" not in genres:
            genres.append("Thriller")
        if g["Music"] == 1 and "Music" not in genres:
            genres.append("Music")
        if g["Horror"] == 1 and "Horror" not in genres:
            genres.append("Horror")
        if g["Documentary"] == 1 and "Documentary" not in genres:
            genres.append("Documentary")
        if g["Mystery"] == 1 and "Mystery" not in genres:
            genres.append("Mystery")
        if g["Western"] == 1 and "Western" not in genres:
            genres.append("Western")
        if g["TV Movie"] == 1 and "TV Movie" not in genres:
            genres.append("TV Movie")
        if g["War"] == 1 and "War" not in genres:
            genres.append("War")
        if g["Foreign"] == 1 and "Foreign" not in genres:
            genres.append("Foreign")
    for g in genres:
        size = 0
        films = mongo.db.films.find({"g": 1}).limit(15)
        for f in films:
            if f["movieId"] not in res and f["movieId"] not in moviesSeen:
                res.append(f["movieId"])
                rec.append(f)
                size +=1;
            if size == 2:
                break
    return render_template("personalHome.html", title="personalHome", films = rec[:15])

```

**Figura 2.** Função filmWatchBased()

Este método começa por reunir da coleção userRatings todos os ids dos filmes avaliados de forma positiva pelo utilizador (rating  $\geq 3$ ). De seguida são reunidos os géneros desses filmes e é feita uma busca na coleção films por esses mesmos géneros. Para cada género são colecionados 2 filmes até um máximo de 15 filmes ou até não haver mais géneros para procurar.

### 2.3 Sistemas baseados em conhecimento

Uma das vantagens destes sistemas é que são particularmente úteis em casos em que não há histórico de filmes ou avaliações, ou cenários de *cold-start*.

**Sistemas baseado em restrições** Neste caso através de uma search bar o utilizador pode fazer uma pesquisa com uma determinada palavra, frase ou expressão que pode pertencer ao título, ao resumo ou até ao género do filme, e são-lhe apresentados filmes que sejam encontrados nessa busca. O método encarregue desta busca chama-se *search()*, de seguida apresenta-se o método e explica-se o seu funcionamento.

```
def search():
    if request.method == 'POST':
        text = request.form['search']
        if mongo.db.films.find({text: 1}).count() > 0:
            films = mongo.db.films.find({text: 1})
            return render_template("personalHome.html", title="personalHome", films = films[:15])
        elif text == "False" or text == "True":
            if mongo.db.films.find({'adult': {'$regex': text}}).count() > 0:
                films = mongo.db.films.find({'title': {'$regex': text}})
                return render_template("personalHome.html", title="personalHome", films = films[:15])
        else:
            films = []
            if mongo.db.films.find({'title': {'$regex': text}}).count() > 0:
                films += mongo.db.films.find({'title': {'$regex': text}})
            elif len(films) < 15 and mongo.db.films.find({'original_title': {'$regex': text}}).count() > 0:
                films += mongo.db.films.find({'original_title': {'$regex': text}})
            if len(films) < 15 and mongo.db.films.find({'tagline': {'$regex': text}}).count() > 0:
                films += mongo.db.films.find({'tagline': {'$regex': text}})
            if len(films) < 15 and mongo.db.films.find({'original_language': {'$regex': text}}).count() > 0:
                films += mongo.db.films.find({'original_language': {'$regex': text}})
            if len(films) == 0:
                flash('We haven\'t found the film you wanted!', 'fail')
                films += top10_avg()
            return render_template("personalHome.html", title="personalHome", films = films)
        else:
            films = filmsColdCases()
            return render_template("personalHome.html", title="personalHome", films = films)
```

Figura 3. Função *search()*

Este método começa por verificar se foi chamado por um método *POST* se não foi retorna o método *filmsColdCases()* descrito na próxima subsecção. Caso tal não ocorra este verificará se se trata de um género e se se tratar retorna os primeiros 15 filmes encontrados por ele, caso não se trate verifica se a palavra inserida foi "True" ou "False", caso tenha sido realiza uma busca única para verificar se se trata de um filme adulto ou não. Se nenhuma das opções acima descritas foi seguida passamos para uma busca por título ou título original, por tagline, por linguagem original e por fim caso não tenhamos resultados suficientes (filmes encontrados == 0), realizamos uma busca pelos 10 melhores filmes e imprimimos uma mensagem a dizer que não encontramos os resultados pretendidos e apresentamos alternativas.

### 2.4 Extras

Esta subsecção serve para apresentar alguns métodos úteis disponibilizados para melhorar a experiência dos utilizador ou o desempenho da aplicação.

**Top 10** Caso o utilizador queira ver filmes dos quais provavelmente irá gostar, pode utilizar o método *top10\_avg()* este método é responsável por retornar os 10 filmes mais bem cotados pelo IMDB. Este método foi criado com o objetivo de poder ser uma forma de recomendar bons filmes dos quais o utilizador possa gostar mitigando as dificuldades do cold-start.

```
def top10_avg():
    films = mongo.db.films.find().sort([("vote_average", -1)])
    res = []
    for f in films:
        if not isinstance(f["title"], int):
            res += f
    return render_template("personalHome.html", title="personalHome", films = res[:10])
```

**Figura 4.** Função top10\_avg()

**New Films** Caso tenhamos um filme novo no sistema não dispomos de muitas formas de o promover, o método *filmsColdCases()* resolve esse problema. Primeiro coleciona os filmes por ordem crescente do número de votos, caso tenham o mesmo número de votos ordena-os por data de lançamento. De seguida retira os primeiros 15 documentos obtidos (cada documento corresponde a um filme) e para cada um aumenta em 1 o número de votos na base de dados, isto permite alterar os filmes apresentados visto que eles são ordenados pelo número de votos que possuem, ou seja o seu número de votos vai aumentando sempre, fazendo com que se por exemplo o documento 16 tivesse 0 votos na próxima vez que o método fosse executado este passaria a ser o primeiro resultado dado que os 15 anteriores agora teriam 1 voto.

```
def filmsColdCases():
    films = mongo.db.films.find().sort([("vote_count", 1), ("release_date", -1)])
    res = []
    for f in films:
        if not isinstance(f["title"], int):
            res += mongo.db.films.find(f)
    for f in res[:15]:
        mongo.db.films.update_one(f, {"$inc": { "vote_count": 1 }} )
    return res[:15]
```

**Figura 5.** Função filmsColdCases()

**Best Rated Films by Users** Se o utilizador quiser ver o que outros utilizadores acham interessante, pode fazê-lo graças ao método *memBased()*, este método é útil em situações em que por exemplo o utilizador nunca tenha visto filmes de Drama na plataforma embora goste deles, desta forma caso outros utilizadores tenham visto e dado boa pontuação a filmes deste genero o utilizador pode obter uma excelente recomendação e ao ver esse filme cria uma ponte de ligação entre os seus gostos e os dos utilizadores usados para a recomendação.

```
def memBased():
    best15 = mongo.db.userRating.find({"movieId": {"$lt": 1000}}, {"movieId": 1, "rating": 1, "_id": 0}).sort([("rating", -1)]).limit(15)
    ratings = []
    for movie in best15:
        ratings.append(movie["rating"])
    return render_template("personalHome.html", title="personalHome", films = best15[:15], ratings = ratings)
```

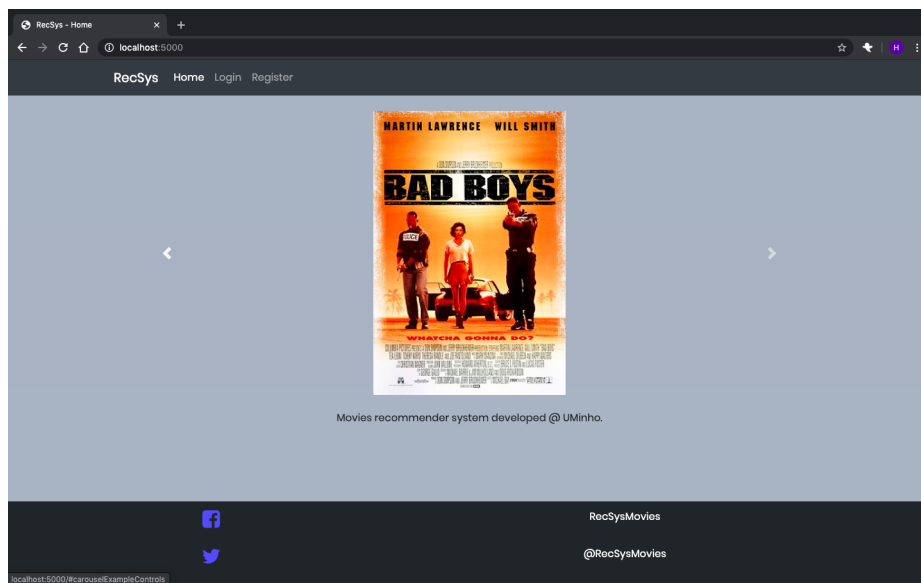
**Figura 6.** Função memBased()



## 3 Aplicação

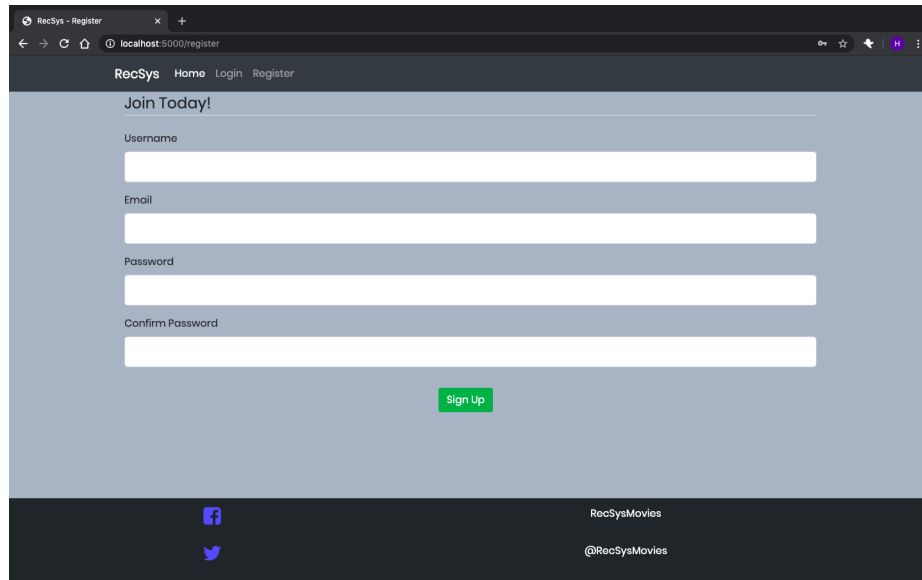
### 3.1 Funcionamento

Ao abrir a aplicação na página web podemos ver a página mostrada abaixo. Esta mostra alguns filmes presentes no nosso sistema de recomendação, que alternam entre si a cada 4 segundos desde que o utilizador não tenha o rato em cima de um. Esta página permite fazer o registo de um novo utilizador ou que este faça o login, caso já tenha conta.



**Figura 7.** Homepage RecSys

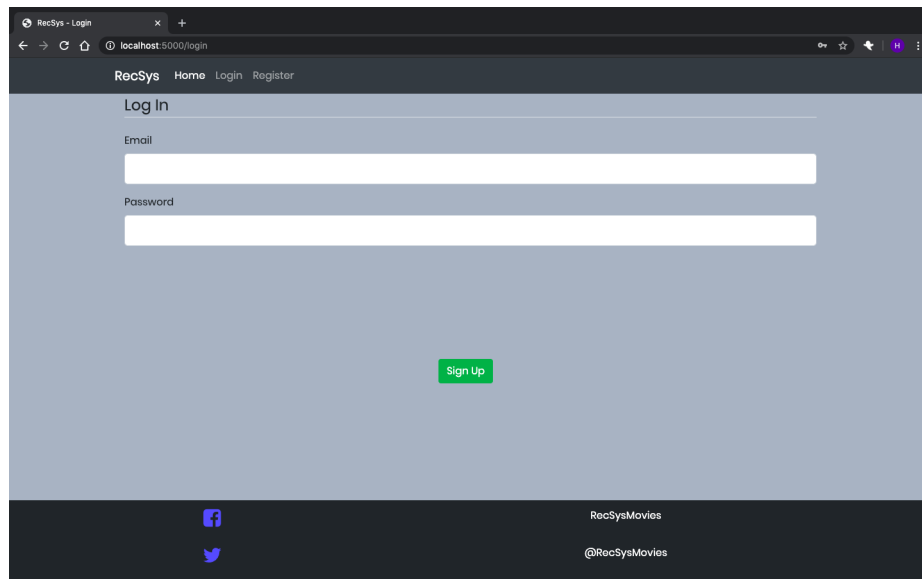
Caso não tenhamos conta temos de fazer o registo. Neste temos de inserir um nome de utilizador, um email válido e uma password. Ao inserir na base de dados um id é gerado para o utilizador e inserido no campo id do documento na coleção users.



A screenshot of a web browser showing the 'RecSys - Register' page. The browser's address bar displays 'localhost:5000/register'. The page has a dark header with 'RecSys' and navigation links 'Home', 'Login', and 'Register'. The main content area is light blue and features the heading 'Join Today!'. Below this are four white input fields labeled 'Username', 'Email', 'Password', and 'Confirm Password'. A green 'Sign Up' button is positioned below the fields. The footer is dark and includes social media icons for Facebook and Twitter, along with the text 'RecSysMovies' and '@RecSysMovies'.

**Figura 8.** Register Page RecSys

Caso já tenhamos conta devemos efetuar o login inserindo o email e a password.



A screenshot of a web browser showing the 'RecSys - Login' page. The browser's address bar displays 'localhost:5000/login'. The page has a dark header with 'RecSys' and navigation links 'Home', 'Login', and 'Register'. The main content area is light blue and features the heading 'Log In'. Below this are two white input fields labeled 'Email' and 'Password'. A green 'Sign Up' button is positioned below the fields. The footer is dark and includes social media icons for Facebook and Twitter, along with the text 'RecSysMovies' and '@RecSysMovies'.

**Figura 9.** Login Page RecSys

Quando um utilizador efetua o login com sucesso na sua conta pode ver os filmes que foram recentemente adicionados à nossa coleção (são os filmes com menos votos da nossa base de dados e são ordenados por data caso o número de votos coincidam). Este sistema funciona graças ao sistema implementado na função `filmsColdCases()` que promove os filmes que não têm como ser recomendados de outra forma por falta de avaliações de utilizadores.

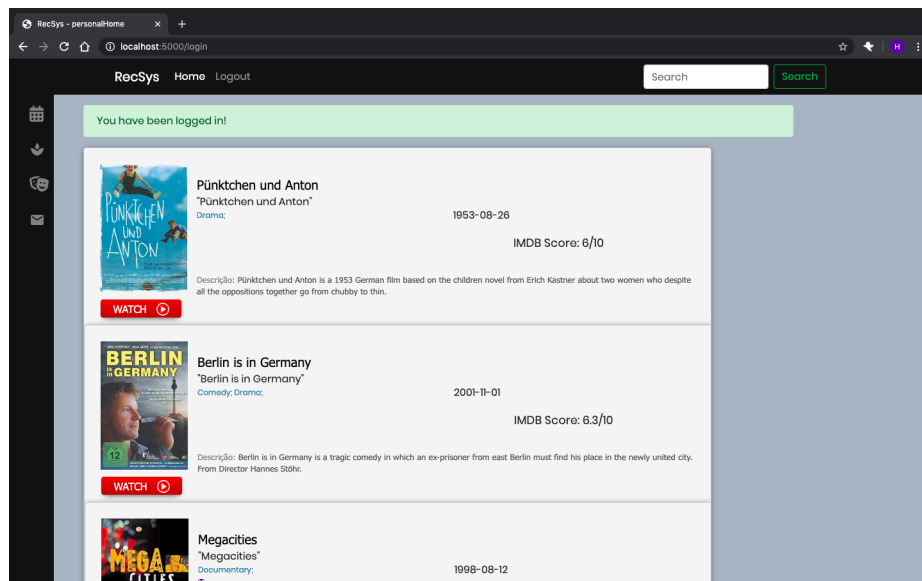
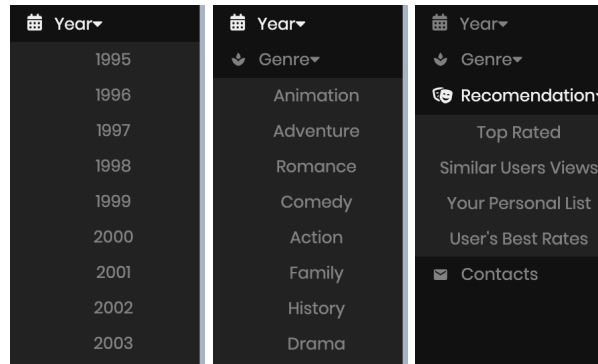


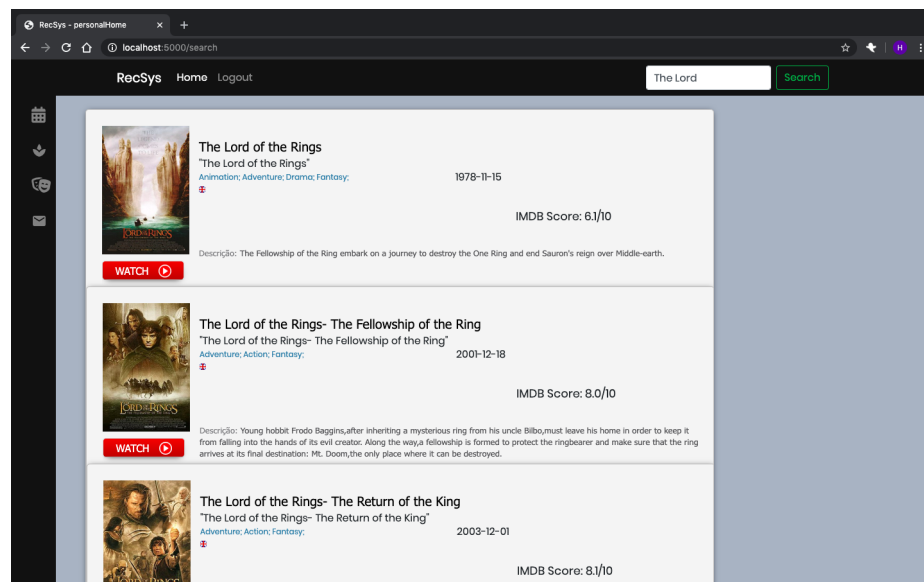
Figura 10. Start Page RecSys

Adicionalmente é possível através desta página pesquisar filmes por ano de lançamento (apenas são mostrados anos de filmes presentes na base de dados), por gênero e por um dos sistemas de recomendação disponibilizados.



**Figura 11.** Year, Genre and Recomendation Search options RecSys

Uma alternativa para procurar filmes é utilizar a *search bar* disponibilizada, onde o utilizador introduz uma palavra, frase, ou expressão e todos os filmes com essa palavra/frase/expressão associada são apresentados.



**Figura 12.** Search Bar Results Page RecSys

Após visualizar o filme o utilizador poderá atribuir uma pontuação (1 a 5 estrelas). Note-se que o botão submit fica bloqueado até uma classificação ser escolhida.

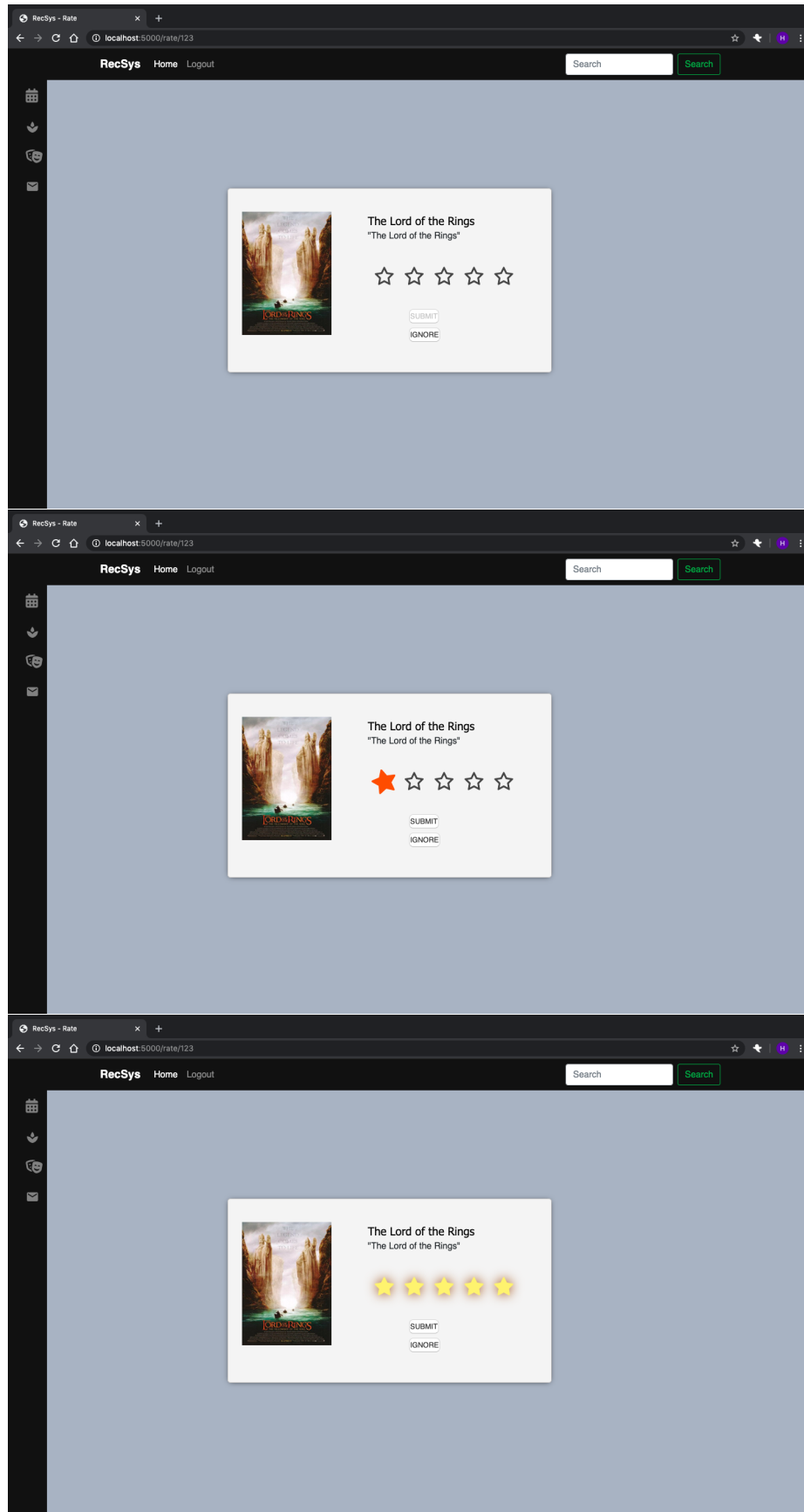
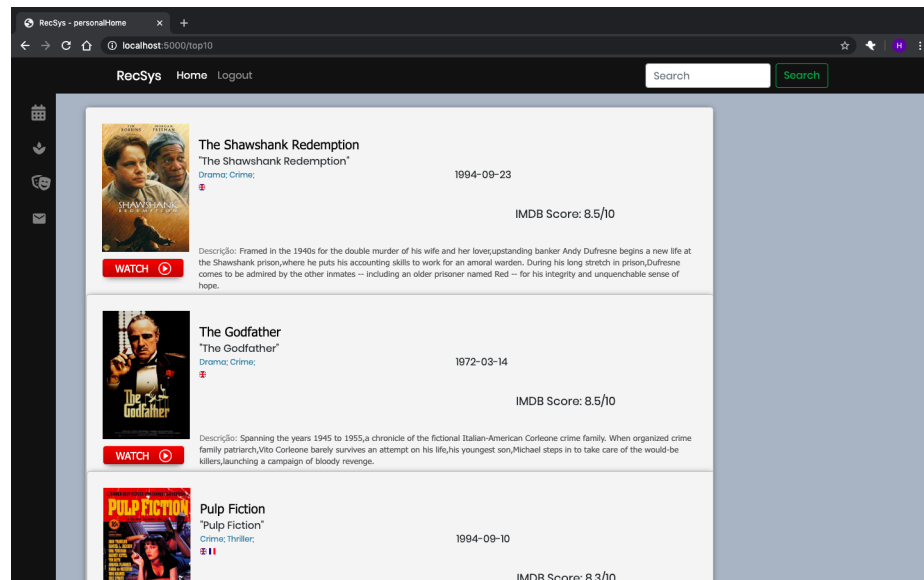


Figura 13. Rating Page RecSys

Sempre que um utilizador atribui uma pontuação a um filme é adicionada essa informação à base de dados na coleção `userRating`. Por exemplo, se um utilizador `x` que viu um filme `y` e atribuiu uma pontuação `z`, esta a informação fica guardada na forma `"_id": ObjectId(...), "userId": x, "movieId": y, "rating": z, "timestamp": ....`

Se escolhermos o sistema de recomendação *"Top Rated"*, obteremos os 10 melhores filmes segundo a IMDB ordenados por ordem decrescente de nota.



**Figura 14.** Top Rated Page RecSys

Se escolhermos o sistema de recomendação *"Similar User's Views"*, obtemos os filmes mais bem cotados pelos `X` utilizadores mais semelhantes. Este algoritmo funciona com base no método `colab_filtering()` descrito anteriormente.

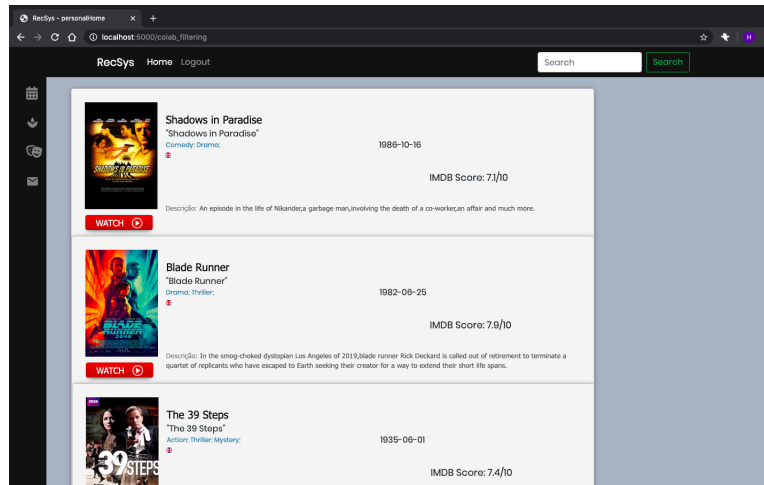


Figura 15. Similar User's Views Page RecSys

Se escolhermos o sistema de recomendação *"Your Personal List"* obtemos uma lista de filmes baseado nos gêneros dos filmes vistos pelo utilizador esta lista resulta da aplicação do método `filmWatchBased()`.

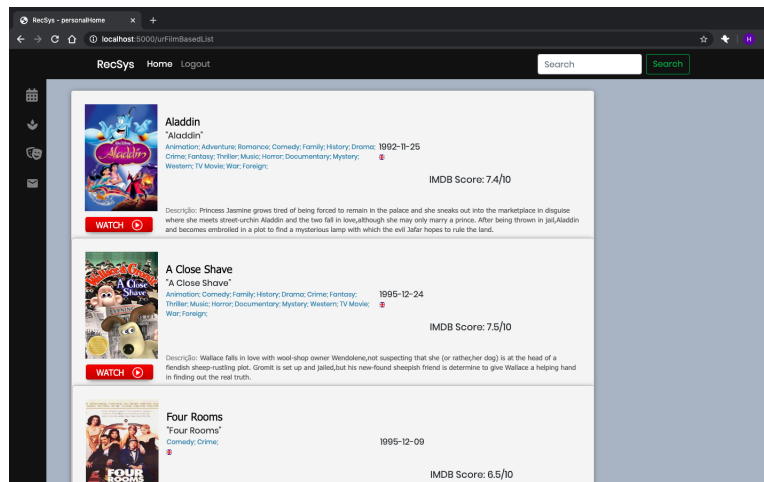
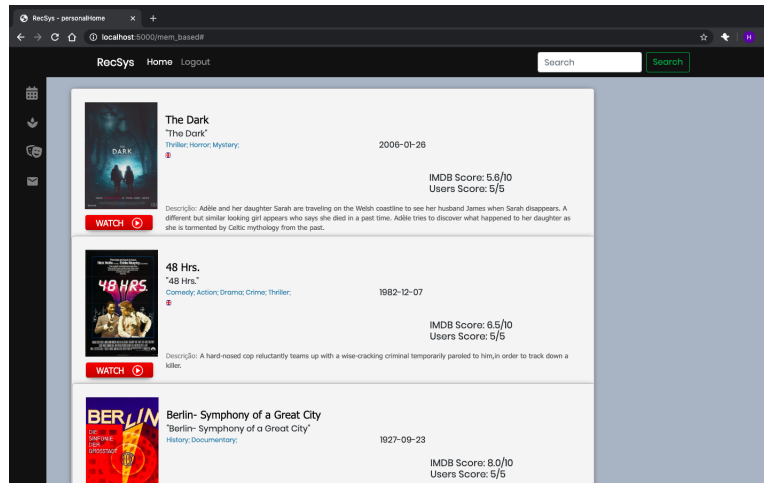


Figura 16. Your Personal List Page RecSys

Por fim se escolhermos o sistema de recomendação *"User's Best Rates"* obtemos uma lista dos filmes mais bem cotados por todos os utilizadores do sistema.



**Figura 17.** User's Best Rates Page RecSys



## 4 Conclusão

No nosso sistema de recomendação apresentado são usadas várias abordagens, com o intuito de fornecer recomendações baseadas nos gêneros dos filmes, bem como nos ratings atribuídos a cada filme. Se um utilizador classificar um filme de um gênero em particular, recomenda-se filmes que contenham gêneros semelhantes. O sistema de recomendação pode ser testado na nossa aplicação RecSysMovies. No entanto, no nosso trabalho foram utilizados vários atributos, considerando-se assim um sistema bastante complexo. Todas as abordagens mencionadas ao longo deste trabalho têm os seus próprios benefícios e problemas, mas ainda não estão no nível adequado para resolver todos os problemas relacionados, por exemplo, à segurança. Um dos trabalhos futuros seria melhorar a nossa aplicação bem como o sistema de Recomendação implementado...