Giải thích chi tiết mã nguồn Flask

---

1. Thiết lập và khởi tạo

Nhập thư viện và cấu hình môi trường

- **Thư viện sử dụng**:
  - `dotenv`: Dùng để tải `API_KEY` từ file `.env` (bảo mật thông tin API).
  - `pandas`: Dùng để đọc và xử lý dữ liệu từ file CSV (chứa thông tin phim).
  - `Flask`: Framework web dùng để xây dựng ứng dụng.
- **Cấu hình môi trường**:

```
load_dotenv()
api_key = os.getenv("API_KEY")
```

Lệnh này nạp biến môi trường từ file `.env` để lấy `API_KEY`. Biến này dùng để truy cập API của TMDb (The Movie Database).

Nạp dữ liệu và xử lý

- **Nạp dữ liệu từ file `data.csv`**:

```
df_movies = pd.read_csv('/home/cp/project/webs/data/data.csv', encoding='utf-8')
```

Dữ liệu phim được lưu trong file CSV, bao gồm thông tin về tên phim, đánh giá, thể loại, diễn viên,...

- **Xử lý dữ liệu**:

```
df_movies['vote_average'] = pd.to_numeric(df_movies['vote_average'], errors='coerce')
df_movies['vote_count'] = pd.to_numeric(df_movies['vote_count'], errors='coerce')
df_movies['runtime'] = pd.to_numeric(df_movies['runtime'], errors='coerce')
```

Các cột `vote_average`, `vote_count`, `runtime` (điểm đánh giá, số lượt đánh giá, thời lượng phim) được chuyển đổi về dạng số (nếu có lỗi, giá trị sẽ thành `NaN`).

- **Chuyển đổi cột dạng chuỗi thành danh sách**:

```
df_movies['genres_list'] = df_movies['genres_list'].apply(eval)
df_movies['actor'] = df_movies['actor'].apply(eval)
```

Các cột `genres_list` (danh sách thể loại) và `actor` (diễn viên) được chuyển từ dạng chuỗi JSON sang danh sách Python.

- **Tạo danh sách tiêu đề phim**:

```
movie_titles = df_movies['title'].tolist()
```

Dùng để hiển thị trong thanh tìm kiếm.

---

2. Lấy ảnh poster

Hàm `fetch_poster()`

- Hàm này dùng API của TMDb để lấy ảnh poster của phim:

```
def fetch_poster(movie_id):
    url = f"https://api.themoviedb.org/3/movie/{movie_id}?api_key={api_key}&language=en-US"
    response = requests.get(url).json()
    poster_path = response.get('poster_path')
    return f"https://image.tmdb.org/t/p/w500/{poster_path}" if poster_path else "https://via
```

- Tạo URL API dựa trên `movie_id` và `API_KEY`.
- Gửi yêu cầu đến API và nhận phản hồi (JSON).
- Nếu có `poster_path`, tạo URL ảnh từ TMDb. Nếu không, trả về ảnh placeholder.

Bộ nhớ đệm (cache)

- **Tối ưu hóa** bằng cách lưu URL ảnh đã lấy:

```
poster_cache = {}
if movie_id in poster_cache:
    return poster_cache[movie_id]
```

Điều này giúp tránh việc gọi API nhiều lần cho cùng một phim, giảm thời gian và chi phí.

---

3. Tìm kiếm phim

Hàm `search_movies()`

- Tìm phim dựa trên tiêu đề và điểm đánh giá tối thiểu:

```
matching_movies = df_movies[
    df_movies['title'].str.contains(query, case=False, na=False) &
    (df_movies['vote_average'] >= min_rating)
]
```

- Tìm phim có tiêu đề chứa `query` (không phân biệt chữ hoa/thường).
- Lọc các phim có điểm đánh giá (`vote_average`) lớn hơn hoặc bằng `min_rating`.
- Trả về danh sách kết quả gồm:
  - `title` (tên phim).
  - `poster_path` (URL poster).

- `vote_average` (điểm đánh giá).

---

4. Định tuyến (Routes) của Flask

Trang chủ (/)

- Hiển thị trang tìm kiếm phim:

```
@app.route('/')
def home():
    return render_template('index.html', movie_list=movie_titles, search_query="", rating_fi
```

- Dữ liệu truyền vào:
  - `movie_list`: Danh sách tiêu đề phim (hiển thị dưới dạng datalist).
  - `search_query`: Câu truy vấn tìm kiếm (mặc định rỗng).
  - `rating_filter`: Lọc theo điểm đánh giá (mặc định là 0).

Tìm kiếm phim (`/recommend`)

- Xử lý form tìm kiếm:

```
@app.route('/recommend', methods=['POST'])
def recommend():
    movie_name = request.form['movie_name']
    min_rating = float(request.form['rating'])
    recommendations = search_movies(movie_name, min_rating)
    return render_template(
        'index.html',
        movie_list=movie_titles,
        recommendations=recommendations,
        search_query=movie_name,
        rating_filter=min_rating
    )
```

- Lấy dữ liệu từ form:
  - `movie_name`: Tên phim cần tìm.
  - `rating`: Điểm đánh giá tối thiểu.
- Gọi hàm `search_movies()` để tìm phim phù hợp và trả về kết quả.

Trang thành viên (`/members`)

- Hiển thị thông tin các thành viên nhóm:

```
@app.route('/members')
def members():
    team_members = [
        {"MSSV": "22110158", "Hovaten": "Tran Chau Phu"},
        {"MSSV": "22110170", "Hovaten": "Ho Minh Quan"},
```

```
        {"MSSV": "22110123", "Hovaten": "Le Nguyen Duc Nam"},
        {"MSSV": "22110124", "Hovaten": "Le Thi Kim Nga"},
        {"MSSV": "22110155", "Hovaten": "Tran Nguyen Thanh Phong"},
    ]
    return render_template('members.html', members=team_members)
```

Trang tài liệu (`/documents`)

- Liệt kê danh sách tài liệu:

```
@app.route('/documents')
def documents():
    doc_folder = './static/doc'
    files = os.listdir(doc_folder)
    return render_template('documents.html', files=files)
```

Tải tài liệu (`/doc/<filename>`)

- Trả về file PDF:

```
@app.route('/doc/<filename>')
def serve_pdf(filename):
    return send_from_directory('./static/doc', filename)
```

---

5. Chạy ứng dụng

- Ứng dụng được chạy ở chế độ debug:

```
if __name__ == '__main__':
    app.run(debug=True)
```

Chế độ debug giúp tự động làm mới khi thay đổi mã và cung cấp thông tin lỗi chi tiết.

---

6. Gợi ý cải tiến

1. **Bảo mật**:
   - Thay `eval()` bằng `json.loads()` để tránh lỗi bảo mật.
2. **Xử lý lỗi**:
   - Thêm xử lý lỗi khi file CSV không tồn tại hoặc API không phản hồi.
3. **Cải thiện giao diện**:
   - Sử dụng thư viện CSS/JS như Bootstrap để làm đẹp UI.
4. **Thêm bộ lọc nâng cao**:
   - Lọc theo thể loại, diễn viên hoặc thời gian phát hành.