Low-Level Document (LLD)

**Movie Recommendation Application Based on Category**

---

#### \*\*1. System Components\*\*

\*\*1.1 Frontend:\*\*

- \*\*Technology:\*\* Streamlit

- \*\*Components:\*\*

- \*\*Category Selection Widget:\*\* Dropdown menu for users to select movie categories.

- \*\*Recommendations Display Area:\*\* Section to display recommended movies, including title, poster, and description.

\*\*1.2 Backend:\*\*

- \*\*Technology:\*\* Python

- \*\*Components:\*\*

- \*\*API Handler:\*\* Manages requests to the movie database/API.

- \*\*Recommendation Engine:\*\* Applies a basic recommendation algorithm to generate movie suggestions.

- \*\*Data Processing Module:\*\* Handles data extraction, processing, and formatting.

\*\*1.3 Movie Database/API:\*\*

- \*\*API:\*\* TMDb API or a local dataset

- \*\*Components:\*\*

- \*\*Movie Data Fetcher:\*\* Retrieves movie details based on the selected category.

- \*\*Data Parser:\*\* Parses the response data from the API or dataset.

---

#### \*\*2. Frontend Design\*\*

\*\*2.1 Streamlit Interface:\*\*

- \*\*Category Selection Widget:\*\*

- \*\*Component Type:\*\* Dropdown

- \*\*Options:\*\* List of movie categories (e.g., Action, Comedy, Drama)

- \*\*Event Handling:\*\* On selection, sends a request to the backend for recommendations.

- \*\*Recommendations Display Area:\*\*

- \*\*Components:\*\*

- \*\*Movie Title:\*\* Text element showing the movie title.

- \*\*Movie Poster:\*\* Image element displaying the movie poster.

- \*\*Movie Description:\*\* Text element providing a brief description of the movie.

\*\*Example Code:\*\*

```python

import streamlit as st

# Category Selection

categories = ["Action", "Comedy", "Drama"]

selected\_category = st.selectbox("Select Movie Category", categories)

# Display Recommendations

if selected\_category:

recommendations = get\_recommendations(selected\_category)

for movie in recommendations:

st.image(movie['poster'], caption=movie['title'])

st.write(movie['description'])

```

---

#### \*\*3. Backend Design\*\*

\*\*3.1 API Handler:\*\*

- \*\*Function:\*\* Fetches movie data from TMDb API or local dataset based on category.

- \*\*Endpoints:\*\*

- \*\*API URL:\*\* `https://api.themoviedb.org/3/discover/movie`

- \*\*Parameters:\*\* `category`, `api\_key`

- \*\*Example Code:\*\*

```python

import requests

def fetch\_movie\_data(category):

api\_key = "your\_api\_key"

url = "https://api.themoviedb.org/3/discover/movie"

params = {

'api\_key': api\_key,

'with\_genres': category,

}

response = requests.get(url, params=params)

return response.json()

```

\*\*3.2 Recommendation Engine:\*\*

- \*\*Function:\*\* Processes fetched data and applies a basic recommendation algorithm.

- \*\*Algorithm:\*\* Simple filtering based on category.

- \*\*Example Code:\*\*

```python

def get\_recommendations(category):

data = fetch\_movie\_data(category)

recommendations = []

for movie in data['results']:

recommendations.append({

'title': movie['title'],

'poster': f"https://image.tmdb.org/t/p/w500{movie['poster\_path']}",

'description': movie['overview']

})

return recommendations

```

\*\*3.3 Data Processing Module:\*\*

- \*\*Function:\*\* Handles data extraction, formatting, and preparation for display.

- \*\*Example Code:\*\* Integrated within `get\_recommendations` function as shown above.

---

#### \*\*4. Data Flow\*\*

1. \*\*User selects a category\*\* from the Streamlit interface.

2. \*\*Frontend sends a request\*\* to the backend with the selected category.

3. \*\*Backend fetches movie data\*\* from the TMDb API or local dataset based on the category.

4. \*\*Backend processes the data\*\* and generates recommendations.

5. \*\*Recommendations are sent back\*\* to the frontend.

6. \*\*Frontend displays the recommendations\*\* in the interface.

---

#### \*\*5. Error Handling\*\*

- \*\*API Errors:\*\* Handle errors if the TMDb API returns an error (e.g., invalid API key, quota exceeded).

- \*\*Example:\*\* Display a user-friendly error message.

- \*\*Data Processing Errors:\*\* Handle errors during data extraction and processing.

- \*\*Example:\*\* Provide fallback or default recommendations.

\*\*Example Code:\*\*

```python

def fetch\_movie\_data(category):

try:

response = requests.get(url, params=params)

response.raise\_for\_status()

return response.json()

except requests.exceptions.RequestException as e:

print(f"API Request Error: {e}")

return {'results': []}

```

---

#### \*\*6. Testing\*\*

- \*\*Unit Tests:\*\* Test individual components such as API handlers and recommendation algorithms.

- \*\*Integration Tests:\*\* Test the end-to-end functionality, including frontend interaction and backend processing.

\*\*Example Testing Libraries:\*\*

- \*\*Frontend:\*\* Streamlit's testing utilities or custom test scripts.

- \*\*Backend:\*\* `unittest` or `pytest` for Python.

---

#### \*\*7. Deployment\*\*

- \*\*Frontend:\*\* Deploy the Streamlit app on a platform like Streamlit Sharing or Heroku.

- \*\*Backend:\*\* Host the Python backend on a cloud service or local server.

---

Feel free to adjust this document based on specific details of your implementation and additional requirements.