Movie Explorer Group Project Report

INTRODUCTION

This project aims to create a console-based Movie Explorer application. Its main objectives are to help users search for movies and TV shows, view details, and quickly discover trending titles using real data from the TMDb API.

This report follows a clear structure: Introduction, Background, Specifications and Design, Implementation and Execution, Testing and Evaluation, and Conclusion.

BACKGROUND

Many people struggle to remember movies they want to watch or feel overwhelmed by the number of options available on streaming platforms. Our project provides an easy-to-use console application that allows users to search, explore, and make quick decisions about movies and TV shows.

Alongside solving this problem, the project allows our team to apply and practise Python programming, database handling, API integration, testing, and collaborative development.

SPECIFICATIONS AND DESIGN

Technical Requirements

- Python console application
- Local SQL database (to store users, favourites, and history)
- API integration to retrieve movie and TV show information via TMDb
- Core logic for searching, recommendations, and managing favourites/history
- Modular code structure
- Error handling and input validation

Non-Technical Requirements

- User-friendly text-based menus
- Clear instructions for beginners
- Reliable and consistent behaviour
- Data privacy — only essential information stored

Design and Architecture

The system is designed in six main modules, plus tests:

1. **Database & ORM Module (Laila)**
   - SQLite (or diff version) database schema for users, favourites, and history
   - ORM or helper functions for reading/writing data
2. **API Module (Stacey)**
   - Requests to TMDb API
   - Parsing and returning structured movie/TV show information
3. **Core Logic Module (Khadija)**
   - Processing searches and recommendations
   - Managing favourites and history
4. **Utils Module (Nada)**
   - Helper functions for formatting, validation, and console output
   - Using Python libraries like itertools and collections
5. **Main/Run Script (Zara)**
   - User interface and menu navigation
   - Orchestrates calls to all other modules
6. **Tests (Ayesha)**
   - Unit tests for all modules

High-level Architecture Flow

User Input → Validation → Database operations → Core Logic → API call → Output to user

---

IMPLEMENTATION AND EXECUTION

Team Roles & Development Approach

Team Member	**Responsibility**
Laila	Database design, SQLite schema, ORM/helper functions
Stacey	TMDb API client, integration, fetching movie/TV data

Khadija	Core logic for searching, recommendations, and favorites
Zara	Main console script, menu navigation, program flow
Nada	Utility functions, formatting, input validation
Ayesha	Unit tests for all modules

The team will collaborate using GitHub for version control and branch management.

Tools and Libraries

- Python 3
- SQLite (or other version) database
- Requests library for API calls
- JSON for structured API responses
- GitHub for collaboration

Implementation Process

Achievements / Aims

- To create a working SQLite database schema and helper functions
- To build a clean menu-driven main script implement search and recommendation logic
- To successfully integrate API responses
- To write unit tests for key components

Potential Challenges

- Handling inconsistent or missing API data
- Keeping module boundaries clear and avoiding duplication
- Ensuring all modules communicate smoothly
- Formatting console output clearly for users

Key Decisions

- Use lightweight SQLite database to simplify setup (if applicable)
- Keep the app console-based instead of building a GUI
- Build reusable helper functions in the Utils module

TESTING AND EVALUATION

Testing Strategy

- Unit tests for each module (database, API, logic, utils)
- Manual testing of menu navigation and user interaction
- Edge-case input testing (empty fields, invalid searches)
- Validation of API responses

Functional & User Testing

- Ensuring users can search movies/TV shows, view details, and manage favourites
- Testing program flow to verify modules communicate correctly
- Peer testing for usability and clarity of menus

System Limitations

- Console interface only — no GUI or notifications
- API reliability depends on internet connection
- Recommendations and trending data are limited to TMDb's database

---

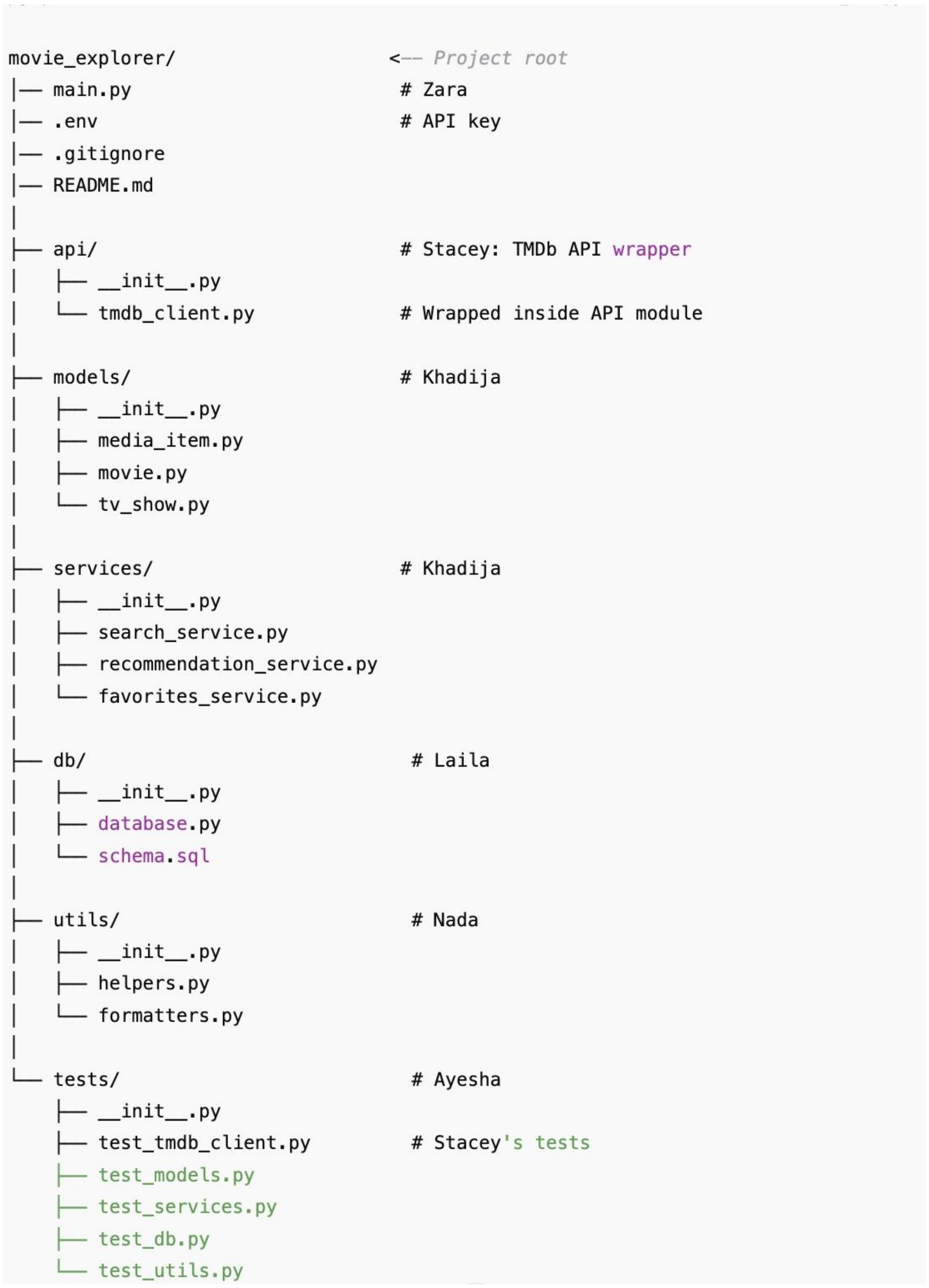
CONCLUSION

The project aims to deliver a functional, console-based Movie Explorer that stores user favourites/history, allows searching and recommendations, and fetches real-time data from the TMDb API.

Each module works together through a clean and structured architecture, and the team demonstrates collaboration and technical growth.

Future enhancements could include a GUI version, more advanced recommendation algorithms, or personalized suggestions based on user history.

GITHUB REPO STRUCTURE



```
movie_explorer/                  <-- Project root
|── main.py                       # Zara
|── .env                          # API key
|── .gitignore
|── README.md
|
├── api/                          # Stacey: TMDb API wrapper
|   ├── __init__.py
|   └── tmdb_client.py            # Wrapped inside API module
|
├── models/                       # Khadija
|   ├── __init__.py
|   ├── media_item.py
|   ├── movie.py
|   └── tv_show.py
|
├── services/                     # Khadija
|   ├── __init__.py
|   ├── search_service.py
|   ├── recommendation_service.py
|   └── favorites_service.py
|
├── db/                            # Laila
|   ├── __init__.py
|   ├── database.py
|   └── schema.sql
|
├── utils/                         # Nada
|   ├── __init__.py
|   ├── helpers.py
|   └── formatters.py
|
└── tests/                         # Ayesha
    ├── __init__.py
    ├── test_tmdb_client.py        # Stacey's tests
    ├── test_models.py
    ├── test_services.py
    ├── test_db.py
    └── test_utils.py
```