## Project Proposal

**Project Name:** Movie Hub
**Project Description:** An interactive web platform for exploring movies, where users can search for films, browse different genres, view detailed movie information, and read reviews.
**Objective**

Provide a user-friendly platform for discovering movies

.Enhance user experience through a fast and engaging interface

.Enable users to rate and comment on movies

.Integrate an API to fetch movie data automatically

## Project Plan (Timeline & Milestones)

- **Week 1:** UI/UX Design (Wireframes, Prototyping, and User Flow).
- **Week 2-3:** Front-End Development (React Components, Styling, and Responsiveness).
- **Week 4:** API Integration (Fetching Movie Data, Handling Requests).
- **Week 5:** Performance Optimization & Enhancements.
- **Week 6:** Final Review & Deployment

## Project Tasks Based on Timeline

**Week 1: UI/UX Design (Wireframes, Prototyping, and User Flow**

**Week 2-3: Front-End Development (React Components, Styling, and Responsiveness)**

Set up the project structure and install dependencies.
Develop reusable React components (Navbar, Movie Cards, Buttons, etc.).

Implement the homepage with movie listings.
Create a responsive movie details page.
Implement the search and filter functionality.
Ensure mobile responsiveness and cross-browser compatibility.

**Week 4: API Integration (Fetching Movie Data, Handling Requests)**

Choose and configure the movie API (e.g., TMDB API).
Fetch and display movie data dynamically.
Implement API error handling and loading states.
Optimize API requests to improve performance.
Test API responses to ensure correct data retrieval.

**Week 5: Performance Optimization & Enhancements**

Optimize image loading and lazy load movie posters.
Minimize API calls and use caching where possible.
Improve website performance and reduce load times.
Enhance animations and user interactions for a smoother experience.
Conduct accessibility testing and UI improvements.

**Week 6: Final Review & Deployment**

Conduct final testing (UI, API, responsiveness, and performance).
Fix any remaining bugs and polish UI components.
Set up hosting and deploy the project (e.g., Vercel, Netlify).
Document the project (README, API usage, and setup instructions).
Share the project link and collect feedback.

## 4. Risk Assessment & Mitigation Plan

| Risk | Potential Impact | Mitigation Strategy |
|---|---|---|
| Slow API Response or Downtime | Poor user experience due to delayed movie data fetching | Use caching techniques and implement a fallback UI in case of API failures. |
| Performance Issues (Slow Loading Pages) | Increased bounce rate and poor user retention | Optimize images, use lazy loading, and minimize unnecessary API calls. |
| Unresolved Bugs Before Deployment | Bad user experience and potential crashes | Implement thorough testing (manual & automated) and conduct final reviews. |
| Hosting or Deployment Issues | The project may not go live on time | Set up deployment (Vercel/Netlify) early and test before final launch. |

## 5. **Key Performance Indicators (KPIs)** – Metrics for Project Success

| **Key** | **Description** | **Target Value** |
|---|---|---|
| Page Load Time | Measures how quickly the website loads. | Less than 2 seconds |
| System Uptime | Ensures the platform is available and functional. | 99.9% uptime |
| User Session Duration | The average time users spend on the platform. | More than 3 minutes |
| User Interactions | Number of searches, reviews, and clicks. | 500+ interactions per month |
| Mobile Responsiveness Score | Ensures the platform works well on mobile devices. | Above 90% on Google Lighthouse |
| API Response Time | Time taken for API requests to fetch movie data. | Under 500ms |

## 2. Literature Review

**Feedback & Evaluation**

- The project will be assessed based on UI/UX design quality, front-end functionality, and API integration efficiency.
- Lecturer’s feedback will focus on usability, performance, and responsiveness.
- Code structure, maintainability, and best practices will also be evaluated.

**Suggested Improvements**

- Enhancing UI/UX design by incorporating animations and micro-interactions.
- Optimizing API calls to reduce latency and improve loading speed.
- Implementing a recommendation system for personalized movie suggestions.
- Adding user authentication for saving favorite movies and watchlists.
- Improving accessibility to ensure a seamless experience for all users.

## Final Grading Criteria

| **Category** | **Weight (%)** | **Description** |
|---|---|---|
| Documentation | 20% | Includes project proposal, literature review, and technical documentation. |
| Implementation | 40% | Quality of front-end development, API integration, and UI responsiveness. |
| Testing & Debugging | 20% | Ensuring functionality, performance, and error handling. |
| Presentation | 20% | Clarity, structure, and effectiveness of the final project presentation. |

## Stakeholder Analysis

**Identifying the key stakeholders involved in the project and their need:**

| **Stakeholder** | **Role** | **Needs & Expectations** |
|---|---|---|
| End Users (Movie Enthusiasts | Visitors using the platform to explore movies. | A user-friendly interface, fast search functionality, and accurate movie data. |
| Project Team (Developers & Designers | Responsible for designing and developing the system. | Clear requirements, efficient API integration, and smooth UI/UX. |
| System Administrator | Maintains system stability and updates. | Secure access, uptime monitoring, and performance optimization. |
| API Provider (Movie Database) | External service providing movie data. | Proper API requests, optimized data fetching, and compliance with API limits. |
| Instructor/Supervisor | Evaluates the project. | Well-structured documentation, functionality, and project presentation. |

## User Stories & Use Cases

**Scenarios illustrating how users interact with the system:**

1. **User Story:** As a user, I want to search for movies by title, genre, or release date so that I can find what interests me.
   - **Use Case:** The user enters a search query, the system fetches movie data, and displays the results.

2. **User Story**: As a user, I want to view detailed information about a movie so that I can learn about its cast, ratings, and synopsis.
   - **Use Case:** The user clicks on a movie, and the system displays details fetched from the API.

3. **User Story**: As a user, I want to add movies to my favorites list so that I can easily access them later.
   - **Use Case:** The user clicks a "Favorite" button, and the system stores the movie in a local database or user account.

4. **User Story**: As a developer, I want to integrate a movie API so that the system retrieves real-time data.
   - **Use Case:** The system sends requests to the API and handles responses efficiently.

---

## Functional Requirements

**List of core features and functionalities:**

- **Movie Search**: Users can search for movies by title, genre, or release date.
- **Movie Details Page**: Displays movie synopsis, cast, ratings, and reviews.
- **Favorites List**: Users can save movies to their favorites for later reference.
- **Responsive Design**: The platform should be accessible on mobile, tablet, and desktop.
- **API Integration:** Fetch movie data from an external API and display it dynamically.
- **Filtering & Sorting:** Users can filter movies by genre, rating, or popularity.

## Non-functional Requirements

**Performance, security, usability, and reliability criteria:**

| **Category** | **Requirement** |
|---|---|
| Performance | Page load time must be under 2 seconds. API response time should be less than 500ms. |
| Security | Secure API requests, prevent unauthorized access to user data. |
| Usability | Intuitive UI with smooth navigation and accessible design. |
| Reliability | Ensure 99.9% uptime and handle API rate limits efficiently**.** |

## System Analysis & Design

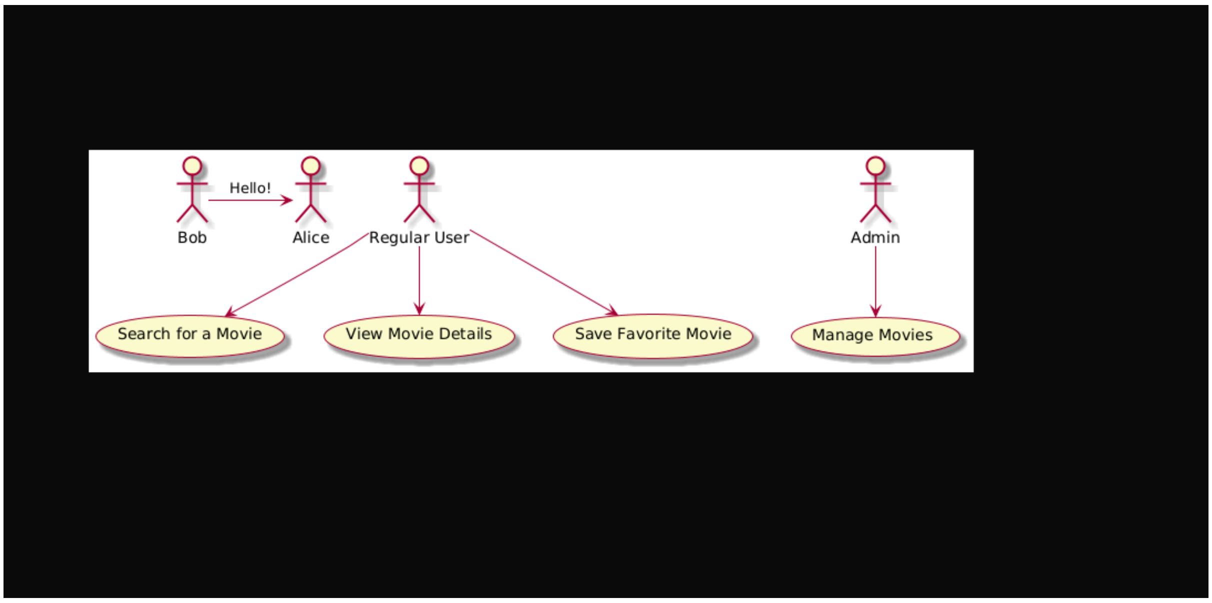
**Problem Statement:**

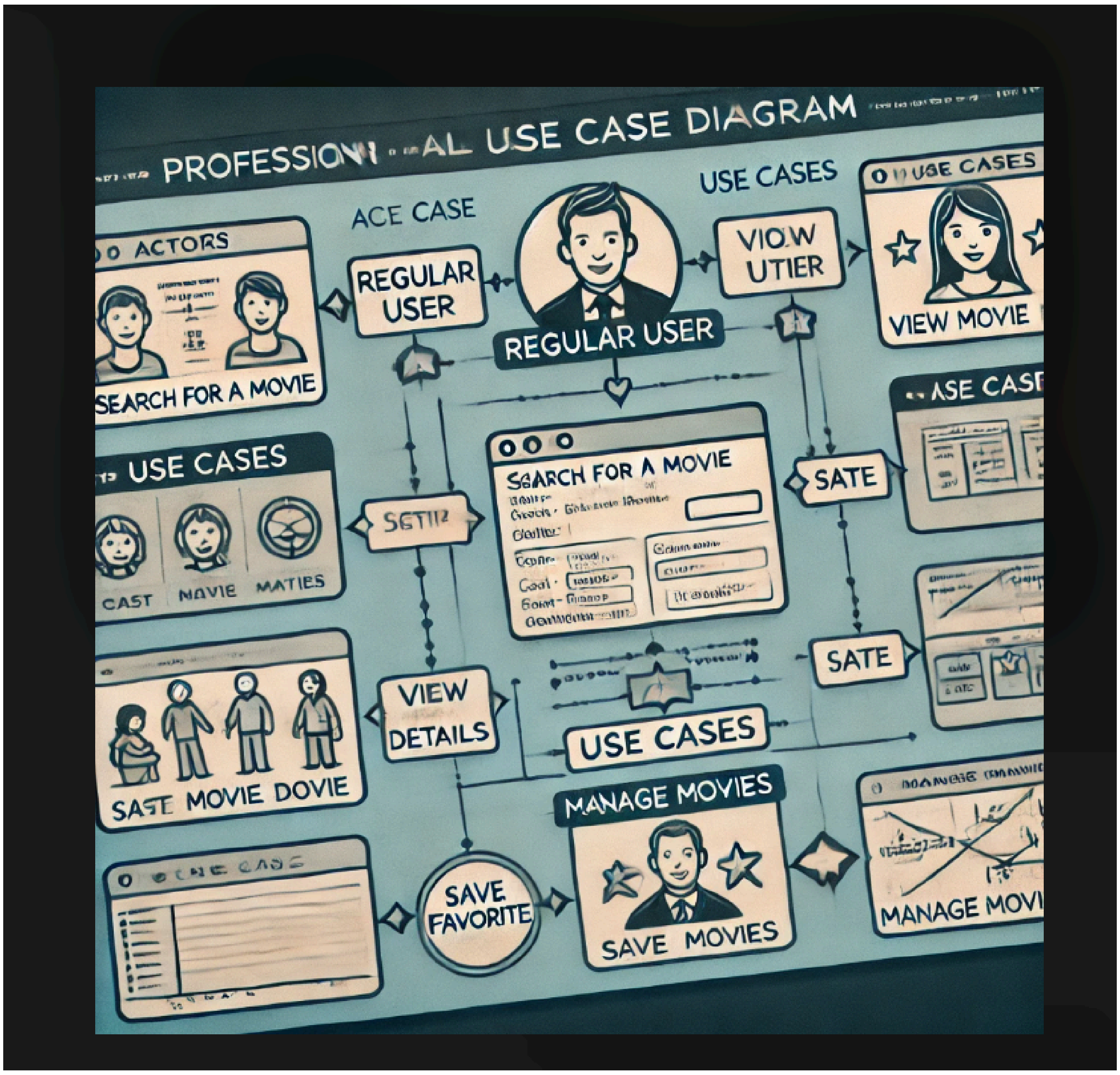
Users face difficulties in finding accurate and quick information about movies, including ratings, cast, and release dates. This project aims to develop an interactive platform that allows users to search for movies easily and retrieve reliable data through an external API.

**Objectives:**

Provide a smooth user experience for movie searching.
Display movie details in an organized and fast manner.
Support a "favorites" feature for saving movies.
Optimize performance and ensure fast data retrieval

## 2. Use Case Diagram & Descriptions





**Annotations on the image:**

- **Actors:**
  - Regular User
  - Admin
- **Use Cases:**

  Search for a Movie – The user searches for a movie, and the system displays relevant results.

  - View Movie Details – The user selects a movie to view detailed information.

    - Save Favorite Movie – The user adds a movie to their favorites list.

## 3. Functional & Non-Functional Requirements

**Functional Requirements:**
Users can search for movies and view details.
Filter movies by genre or rating.
Save favorite movies for later access.

**Non-Functional Requirements:**
Fast performance (data loading within 2 seconds).
Responsive design for all devices.
Secure user data and prevent unauthorized access

## 4. Software Architecture

- Define the **architecture style**, such as **MVC (Model-View-Controller)**.
- Describe **system components**, including:
    - **Front-End:** Developed using **React** for UI design.
    - **API Layer:** Fetching movie data from an **external API**.
    - **Database (if applicable):** Storing user preferences (e.g., favorite movies).

## Database Design & Data Modeling

- **ER Diagram (Entity-Relationship Diagram):**
  - **Represents the relationship between entities like Users, Movies, and Favorites.**
- **Logical & Physical Schema:**
  - **Define tables, attributes, primary keys, and ensure normalization to avoid data redundancy.**

**Entity-Relationship Diagram (ERD) for Movie Project**

**Entities & Attributes:**

1. **Users**
   - `user_id` (Primary Key)
   - `name`
   - `email`
   - `password`
2. **Movies**
   - `movie_id` (Primary Key)
   - `title`
   - `genre`
   - `release_year`
   - `rating`
3. **Favorites**
   - `favorite_id` (Primary Key)
   - `user_id` (Foreign Key → Users)
   - `movie_id` (Foreign Key → Movies)

**Relationships:**

- **One-to-Many:** A **User** can save multiple **Movies** as favorites.

- **Many-to-Many:** A **Movie** can be favorited by multiple **Users** (managed through the `Favorites` table).

**Logical & Physical Schema:**

- **Tables:**
  - `Users(user_id, name, email, password)`
  - `Movies(movie_id, title, genre, release_year, rating)`
  - `Favorites(favorite_id, user_id [FK], movie_id [FK])`
- **Normalization:**
  - Ensured **1st Normal Form (1NF)**: No repeating groups.
  - Ensured **2nd Normal Form (2NF)**: All non-key attributes depend on the primary key.
  - Ensured **3rd Normal Form (3NF)**: No transitive dependencies.

# Data Flow & System Behavior

**1. Data Flow Diagram (DFD)**

A Data Flow Diagram (DFD) illustrates how data moves through a system, highlighting the processes, data stores, and external entities involved.

**Key Components:**

- **Processes:** Activities that transform data (e.g., "Search Movies," "Display Results").
- **Data Stores:** Repositories where data is held (e.g., "Movies Database").
- **External Entities:** Actors interacting with the system (e.g., "User").
- **Data Flows:** Arrows showing the movement of data between components.

**Example:**

1. **User Interaction:**
   - The **User** inputs a movie search query.
   - This input is sent to the **Search Movies** process.
2. **Processing:**
   - The **Search Movies** process queries the **Movies Database**.
   - Relevant movie data is retrieved.
3. **Output:**
   - The **Display Results** process presents the retrieved movies to the **User**.

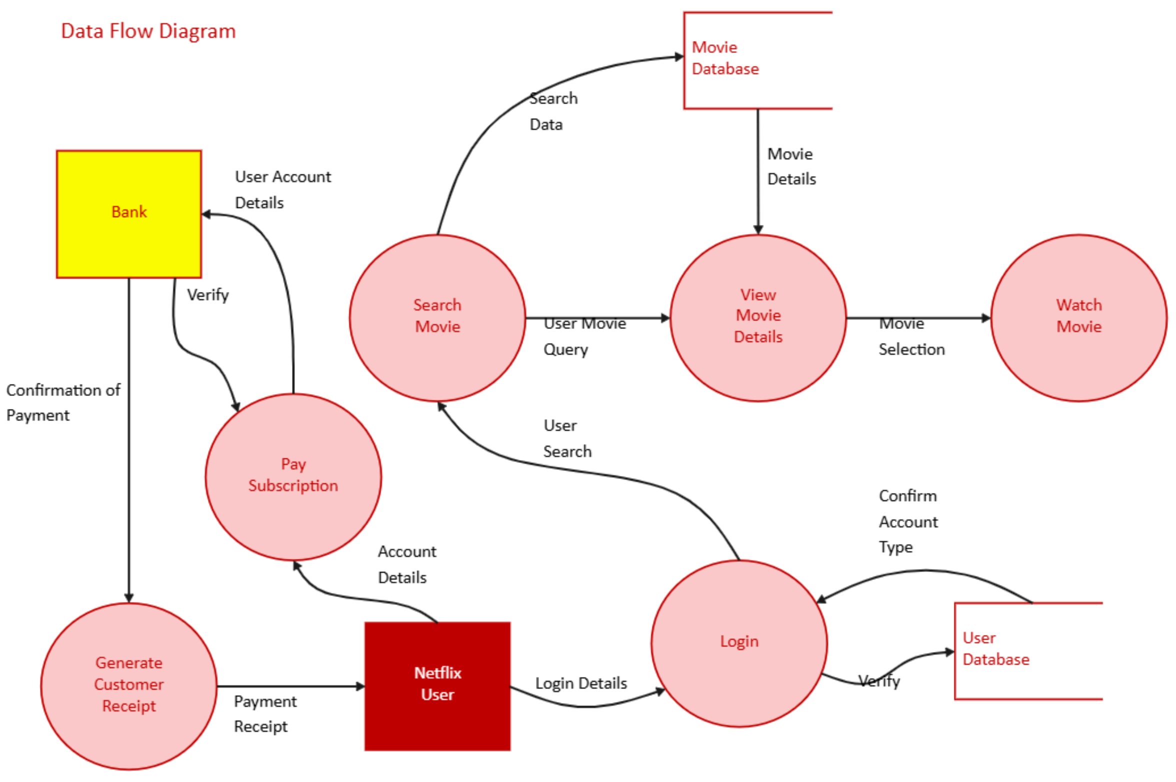
This flow ensures users can efficiently search and view movie details.

#### 2. Sequence Diagram

A Sequence Diagram visualizes the sequence of interactions between objects or components over time, particularly useful for detailing specific functionalities.

**Example: Handling a Movie Search Request**

1. **User Initiates Search:**
   - The **User** enters a movie title into the search interface.
2. **System Processes Request:**
   - The **Search Controller** receives the input and forwards it to the **Movie Service**.
   - The **Movie Service** queries the **Movies Database** for matches.
3. **Results Returned:**
   - The **Movies Database** sends the matching results back to the **Movie Service**.
   - The **Movie Service** processes the data and returns it to the **Search Controller**.
   - The **Search Controller** then displays the results to the **User**.



API Documentation

**Movie APIs** that you can use for your project:

1. **OMDb API (Open Movie Database API)**
   - **Description:** A RESTful web service providing movie information, including user-contributed content and images.
   - **Website:** [https://www.omdbapi.com/](https://www.omdbapi.com/)
2. **TMDb API (The Movie Database API)**
   - **Description:** Offers a vast collection of movie-related data, including movie names, actors, directors, ratings, and posters.
   - **Website:** https://www.themoviedb.org/documentation/api
3. **IMDb API**

- **Description:** Provides access to IMDb data, including movie details, actors, directors, ratings, and images.
- **Website:** https://developer.imdb.com/

4. **Movie Finder API**
   - **Description:** A powerful tool that allows access to an extensive database of movies, including details and related information.
   - **Website:** https://zylalabs.com/api-marketplace/search/movie%2Bfinder%2Bapi/970