رواد مصر الرقمية

وزارة الاتصـــــالات
وتكنولوجيا المعلومات

Movie Recommendation React App

YAT LEARNING SOLUTIONS

Team names

1	ziad Diaa Ezzat
2	Yomna khaled mohamed
3	Sara Mahmoud mohamed
4	Shrouk mohamed rashad
5	Yousef mamdoh fangary

Project description……………………………………………………………….4

Project planning and management…………………………………………..5

Project proposal…………………………………………………………5

Project plan…………………………………………………………………6

Task assignment and roles………………………………………………7

Risk assessment and mitigation plan……………………………….8

Literature Review……………………………………………………………….8

Requirements gathering……………………………………………………9

Stakeholder analysis…………………………………………………….9

User stories and use cases……………………………………………….10

Functional and non functional requirements……………………….11

System analysis and design ……………………………………………….13

Data flow and system behavior…………………………………………14

Ui Ux Design and prototyping…………………………………………..16

Movie Recommendation App using React

This project aims to develop an interactive Movie Recommendation App using React. The app fetches real-time movie data from a third-party API (e.g., The Movie Database API) and provides users with personalized movie recommendations based on genres, trends, and preferences. Key features include search functionality, filtering options (by genre, rating, year), movie details display, and pagination for easy browsing. The project utilizes React for UI development, Redux for state management, and NodeJS for backend integration. The final app will be fully responsive, ensuring a seamless experience across different device

1. Project Planning & Management

1.1 Project Proposal

Overview

The Movie Recommendation App aims to provide users with movie suggestions based on genres, trends, and personal preferences. Built using React, the app will fetch real-time movie data from a third-party API like The Movie Database API, ensuring an up-to-date and engaging experience.

Objectives

The main objective of this project is to create an interactive and user-friendly platform where users can search for movies, filter results by genre, rating, and release year, and browse recommendations effortlessly. The application will leverage Redux for state management, ensuring smooth data handling and updates.

Scope

The project will focus on developing a fully functional web application with a well-structured UI, advanced search and filtering options, and robust API integration. It will not include features like user authentication or a custom database, as it relies on an external API for movie data.

1.2 Project Plan

Week	Task	Time	Milestone	Deliverable	Resources
Week 1	Setup React & NodeJS, create wireframes, basic layout.	7 days	Initial setup complete.	Project setup, wireframes, basic UI.	React, NodeJS, HTML, CSS, JavaScript, Figma.
Week 2	Integrate API, add search & filters, setup Redux.	7 days	API & core features working.	Functional API, movie list, filters.	API key, Redux Toolkit.
Week 3	Add pagination, sorting, unit tests, error handling.	7 days	Advanced features complete.	Pagination, sorting, basic tests.	Jest, debugging tools.
Week 4	Improve UI, test, deploy, write documentation.	7 days	Final version ready.	Responsive UI, deployed app, docs.	Heroku/Netlify, testing tools.

1.3 Task Assignment & Roles

	Team Member name	Tasks
1	Ziad Diaa	Sets up the project structure, configures React environment, and manages global state using Redux.
2	Yousef mamdoh	Integrates the movie API, fetches and displays movie data, and handles API errors.
3	Yomma khaled	Develops the homepage, including trending movies, search bar, and category sections.
4	Shrouk mohamed	Builds the movie details page, including movie information, trailers, and similar recommendations.
5	Sara Mahmoud	Implements navigation with React Router, adds animations, and ensures smooth UI performance.

1.4 Risk Assessment & Mitigation Plan

Risk	Solution
API not working	Show an error message and use backup data.
App slows down	Optimize API calls and use lazy loading.
Filters or search not working	Test features properly and fix bugs early.
Navigation issues	Check all links and add a 404 error page.
UI looks bad on some screens	Test on different devices and adjust styles.
Deployment problems	Test the app before deploying and fix any errors.

2. Literature Review:

- **Feedback & Evaluation – Lecturer's assessment of the project**
  "Your project is well-structured, and the UI looks clean and responsive. The API integration is working smoothly, and Redux is well-implemented for state management. The filtering and sorting functionalities enhance the user experience. Good job on including unit testing and handling API errors properly."
- **Suggested Improvements – Areas where the project can be enhanced.**

Improve performance by implementing lazy loading for images and optimizing API calls to reduce redundant requests.

Enhance user engagement by adding a "Favorites" feature where users can save movies they like.

- **Evaluation Criteria (Lecturer's Perspective):**

- **Functionality:** Are all planned features implemented and working as expected?
- **Code Quality:** Is your React, Redux, and API integration well-structured, modular, and maintainable?
- **User Experience (UX):** Is the UI intuitive, responsive, and engaging?
- **Performance & Optimization:** Is the app efficient in handling API calls and rendering movie data?
- **Testing & Error Handling:** Are unit tests and error handling mechanisms properly implemented?
- **Documentation & Deployment:** Is the project well-documented and successfully deployed?

3. Requirements Gathering:

3.1 Stakeholder Analysis – Identifying key stakeholders and their needs

Stakeholder: End Users (Viewers), Lecturer (Evaluator)
Role & Interest: Primary users who browse and search for movies.
Needs & Expectations: - Easy-to-use interface, Search, filters, accurate recommendations, mobile-friendly design, Fast performance, personalized suggestions, engaging UI.

3.2 User Stories & Use Cases – Scenarios illustrating how users interact with the system.

User Stories:

1. **Browsing Movies**
   As a user, I want to browse a list of movies based on different categories (Trending, Popular, etc.), so that I can easily find something to watch.
2. **Searching for a Movie**
   As a user, I want to search for a specific movie by title, so that I can quickly find information about it.
3. **Filtering Movies**
   As a user, I want to filter movies by genre, release year, and rating, so that I can refine my choices based on my preferences.
4. **Viewing Movie Details**

As a user, I want to click on a movie to see its details (description, rating, cast, etc.), so that I can decide if I want to watch it.

Use Cases: (Describing system interactions in detail)

Use Case 1: Browsing Movies

- **Actor**: User
- **Precondition**: The app is loaded successfully, and API data is available.
- **Flow**:
  1. User opens the app.
  2. The system fetches movie data from the API.
  3. The system displays movie categories (Trending, Popular, etc.).
  4. User scrolls through the list to browse movies.
- **Alternate Flow**: If the API request fails, the system shows an error message.

Use Case 2: Searching for a Movie

- **Actor**: User
- **Precondition**: The app is running.
- **Flow**:
  1. User types a movie title in the search bar.
  2. The system sends a request to the API with the search query.
  3. The system displays search results matching the query.
  4. User clicks on a movie to see details.
- **Alternate Flow**: If no results are found, the system shows a message like “No movies found.”

Use Case 3: Filtering Movies by Genre, Year, or Rating

Actor: User

Precondition: The app is running with available movie data.

Flow:

- User selects filter criteria (e.g., "Action" genre, 2023 release, rating above 7).
- The system applies filters and updates the movie list.
- User browses the filtered list.

Alternate Flow: If no movies match the filters, the system shows a message like "No movies found for this selection."

- Functional Requirements – List of features and functionalities.

-**Movie Browsing & Display**

- Display trending, popular, and recommended movies.
- Show movie details (title, description, rating, genre, cast, etc.).

-**Search & Filtering**

- Allow users to search for movies by title.
- Provide filters for genre, release year, and rating.
- Enable sorting by popularity, rating, or release date.

-**API Integration & Data Handling**

- Fetch real-time movie data from a third-party API (e.g., TMDB).
- Implement pagination for efficient movie browsing.
- Handle API errors and display appropriate messages.

- **State Management (Redux)**

- Manage global state for movies, filters, and user preferences.

-**User Interface & Experience**

- Ensure a mobile-responsive design.
- Provide loading indicators and error messages.

-**Testing & Deployment**

- Include unit tests for core functionalities.
- Deploy the app on a hosting platform (Vercel).

3.3 Non-functional Requirements – Performance, security, usability, and reliability criteria

1. Performance

- The system shall load movie data within **2-3 seconds** under normal network conditions.

- The system shall use **lazy loading** for images to improve loading speed.
- The system should implement **pagination** to manage large datasets efficiently.

2. Security

- The system should store API keys in **environmentally variables** to prevent exposure.
- The system shall enforce **HTTPS** for secure data transmission.

3. Usability

- The system should have an **intuitive and user-friendly UI** with a clear navigation structure.
- The system shall provide **loading indicators** when fetching data.
- The system should ensure **mobile responsiveness** across different screen sizes.

4. Reliability & Availability

- The system will be available **99% of the time** (excluding maintenance).
- The system shall provide **graceful error handling** with user-friendly messages.
- The system should handle **network failures** and retry API requests when possible.

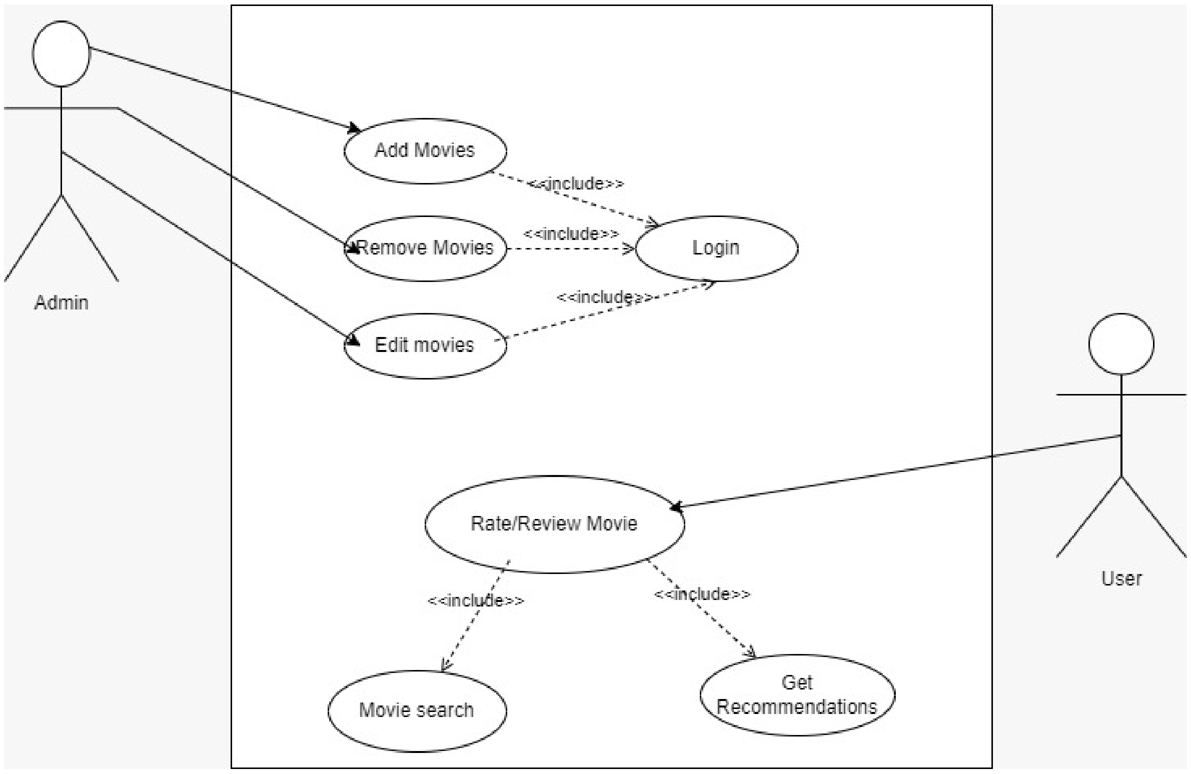
5. Maintainability & Scalability

- The system should use a **modular React component structure** for easy updates.
- The system shall support **future enhancements** like user authentication and watchlists.

4 .System Analysis & Design

4.1 Problem Statement & Objectives – Define the problem being solved and project goals.

4..1.1 use case diagram



4.1.2 Functional & Non-Functional Requirements

Functional Requirements:

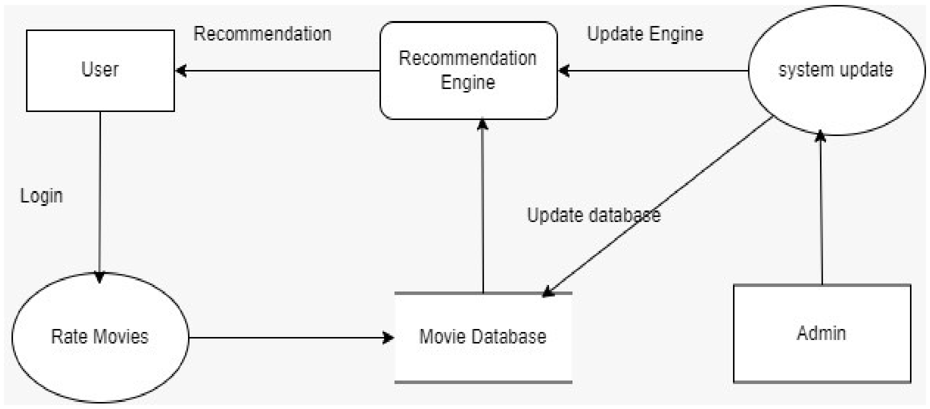
- Fetch and display movie data from an external API.
- Implement search, filtering, and sorting features.
- Allow users to view detailed movie information.
- Provide movie recommendations.

Non-Functional Requirements:

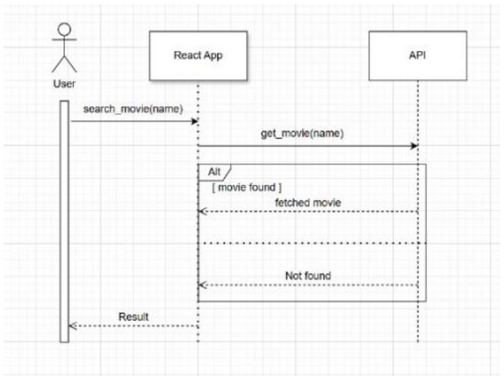
- Performance: Fast response time and smooth navigation.
- Usability: Simple, user-friendly UI.
- Scalability: Support a growing database of movies.
- Reliability: Handle API errors and ensure system stability

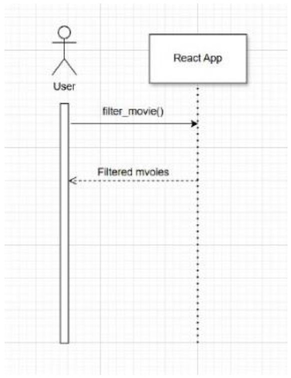
4.2 Data Flow & System Behavior

4.2.1 DFD (Data Flow diagram)

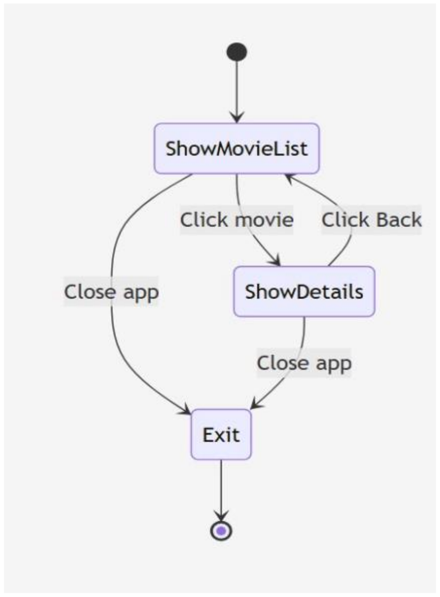


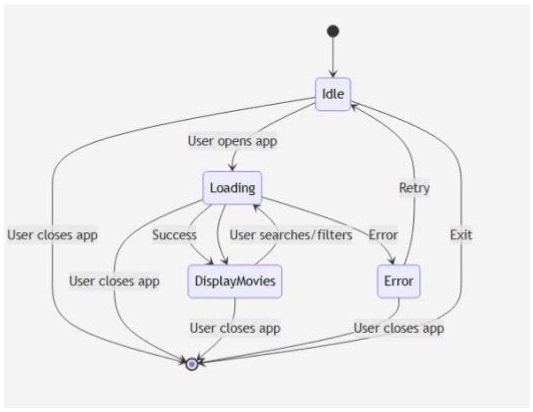
4.2.2 Sequence Diagrams





4.2.3 State Diagrams

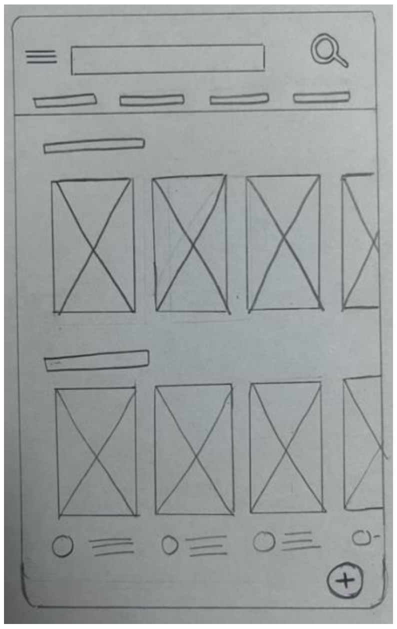


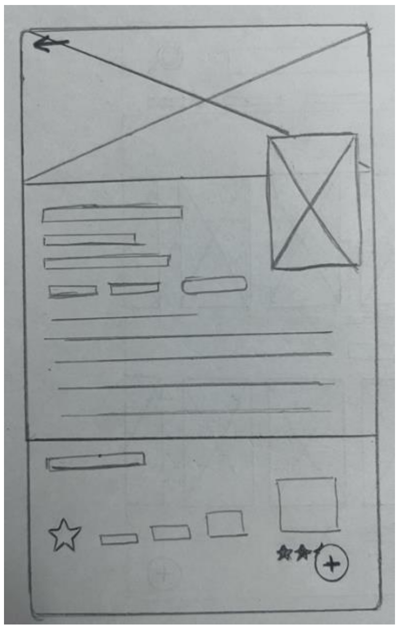


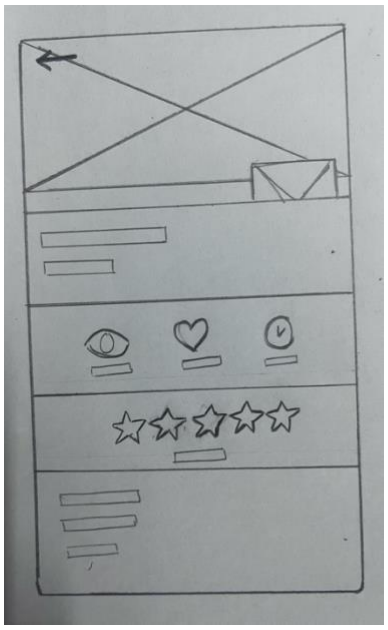
4.3 UI/UX Design & Prototyping

4.3.1 Wireframes & Mockups – Screens and visual representations of the user interface.

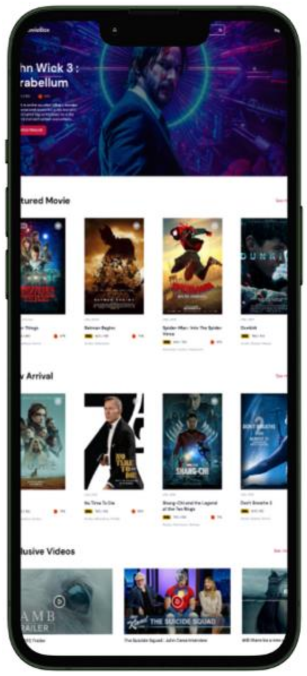
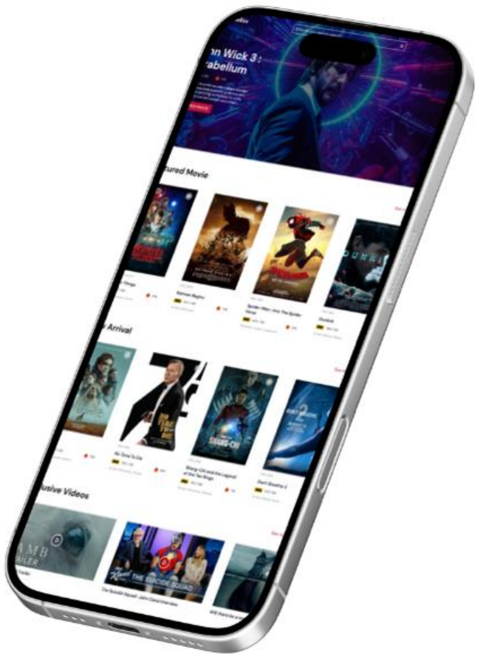
Wireframes:







Mockups:

4.3.2 UI/UX Guidelines – Design principles, color schemes, typography, and accessibility considerations.

1. Design Principles

Consistency: The design maintains a structured layout with a clear hierarchy (e.g., Featured Movies, New Arrivals, Actors).
User Engagement: Large, high-quality images for movie posters enhance visual appeal.
Accessibility: Readable font sizes, high contrast, and well-spaced elements improve usability

2. Typography (Tailwind CSS-Based Sizes)

Different text sizes from text-xs (0.75rem) to text-9xl (8rem) for headings and descriptions.
Font hierarchy ensures readability, e.g.,
text-4xl (MovieBox branding)
text-6xl (Section headings like TV SERIES)
text-sm to text-lg for descriptions

3. Color Scheme

Primary Colors: Cool gray, Red (for accents like CTA buttons), Orange, and Yellow (ratings).
Supporting Colors: Green, Blue, Indigo, Purple, Pink – likely used for different UI elements or accessibility enhancements.

4. Accessibility Considerations

Contrast: The dark background with bright images ensures clarity.
Scalability: Responsive text sizes make it readable on different devices.
Navigation: A clean layout with clear sections enhances user flow.

5. System Deployment & Integration

Frontend:

- **React.js** – For building a dynamic and interactive user interface.
- **Bootstrap** – For responsive and modern styling with utility-first design.
- **JavaScript (ES6+)** – To handle dynamic behavior and interactivity.
- **HTML5 & CSS3** – For structuring and styling the web pages.

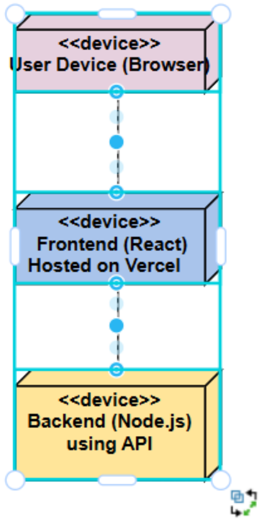
Backend:

- **Node.js** – For handling server-side logic and API requests.
- **Express.js** – As the web framework to manage routing and middleware.

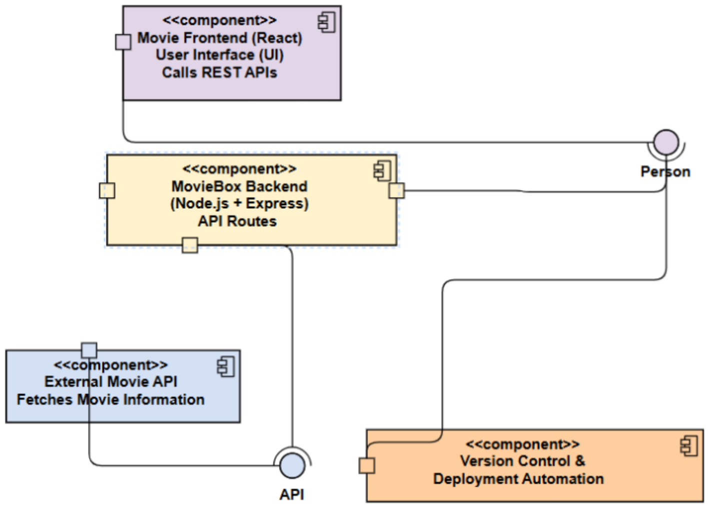
Other Tools & Integrations:

- **GitHub** – For version control and collaboration.

Deployment Diagram – Describes how software components are distributed across hardware.



Component Diagram – Shows high-level system components and their dependencies


<<component>>
Movie Frontend (React)
User Interface (UI)
Calls REST APIs
<<component>>
MovieBox Backend
(Node.js + Express)
API Routes
Person
<<component>>
External Movie API
Fetches Movie Information
API
<<component>>
Version Control &
Deployment Automation