CHAPTER: - 1

INTRODUCTION

1.1 Overview:

The Movie Manager Application is a React.js-based project that allows users to search for movies via an external API (OMDB API) and also manually add their own favorite movies.
This system offers a platform for users to view, manage, and organize their movie collections efficiently.
It combines features of both live search from an API and manual management of user-defined content.

The application provides the following major functionalities:

- Search and Display Movies fetched from OMDB API.
- Add New Movies manually by filling in movie details such as Title, Year, Rating, Description, and Poster URL.
- Delete Movies from both API-fetched and manually added movie lists.
- Custom Movie Management without backend storage, directly handled through the application state.
- Responsive and Simple User Interface designed for ease of use across devices.

By integrating real-time movie search and custom movie addition, the platform ensures that users can create their own personalized movie library, whether based on popular titles or their own creations. The system is interactive, intuitive, and scalable, making it a useful tool for movie enthusiasts who want a simple movie management solution.

1.2 **Problem Definition**:

Today, movie fans often look for platforms where they can search, store, and organize their favorite movies conveniently.
While there are many apps available for browsing movies, most platforms do not offer users the ability to manually add and manage their own movies unless they create an account or subscribe to a service.
There is a need for a simple application that allows:

- Searching movies without login/registration.
- Adding custom movie entries without database complications.
- Managing both API fetched and manually added movies together.
- Deleting movies easily from the list.

Thus, the Movie Manager Application aims to solve this problem by providing a lightweight, easy-to-use solution for searching, adding, and managing movie collections, without the need for a backend server.

1.3 Scope of Proposed System:

The proposed system is developed to fulfill the following needs:

- Movie Search Module:
    - Users can search movies by title using the OMDB API.
    - Complete movie details like Title, Year, Rating, Poster, and Description are displayed.
- Movie Addition Module:
    - Users can add new movies manually with their own details.
    - Poster URL can be customized, or a default image is used if not provided.

- Movie Deletion Module:
    - Movies, whether fetched via API or added manually, can be deleted easily.
- State Management:
    - The system uses React's useState to manage movies without backend storage.
- User Experience (UX):
    - Simple form inputs.
    - Dynamic movie display.
    - Responsive layout for mobile and desktop use.

Additional Scope:

- Customization:
  Users can add movies that may not be available in public databases, allowing for a truly personalized movie list.
- Scalability:
  Though initially built without backend, the system can easily be extended with local storage, databases, or user accounts in the future.
- Flexibility:
  This app structure can be extended to create a full-fledged movie library, a wishlist manager, or a movie recommendation system.

CHAPTER: - 2
REQUIREMENT GATHERING

2.1 **Introduction:**

Requirement gathering is a crucial phase in the software development process where the system's requirements are identified, documented, and agreed upon. This step ensures that the project is built according to the user's needs and expectations. For the Movie Manager Application, the following requirements have been identified:

2.2 **Functional Requirements:**

1. **Movie Search Functionality:**
   - Users should be able to search for movies by title using the OMDB API.
   - Display movie details like title, year, description, rating, and poster image.
2. **Add New Movie Functionality:**
   - Users can manually add movies by providing details such as title, year, rating, description, and poster URL.
   - There should be a default poster image used if the poster URL is not provided.
3. **Delete Movie Functionality:**
   - Users can delete movies from both the fetched movie list and the manually added movie list.
4. **State Management:**
   - The system will utilize React's `useState` to store movie data temporarily in the application state, without the need for backend storage.
5. **User Interface:**
   - The UI should be responsive and simple, providing a seamless experience on both desktop and mobile devices.
   - The UI should be intuitive for adding, viewing, and deleting movies.
6. **Custom Movie Management:**
   - Users should have the option to add and delete their own movies without relying on a database.

2.3 **Non-Functional Requirements:**

1. **Performance:**
   - The application should load quickly, even with multiple movies in the list.
2. **Security:**
   - No sensitive user data is handled, but any future user authentication should be secure.
3. **Usability:**
   - The system should have an intuitive interface with easy-to-use controls.
   - The design should be simple and user-friendly to appeal to movie enthusiasts.
4. **Scalability:**
   - The system should be easily extendable. For example, adding user authentication or integrating a backend for storing movie lists can be done in the future.
5. **Compatibility:**
   - The app should work across all modern browsers and be fully responsive on mobile devices.

2.4 **System Requirements:**

1. **Software Requirements:**
   - Operating System: Windows, macOS, or Linux

    - Programming Language: React.js (JavaScript)
    - Web Browsers: Chrome, Firefox, Edge, Safari
2. **Hardware Requirements:**
    - Processor: Intel Core i3 or equivalent (minimum)
    - RAM: 4 GB (minimum)
    - Storage: 100 MB for app files

2.5 **Data Requirements:**

- The data for the movie list will be fetched from the OMDB API, which provides details such as title, year, rating, poster image, and description.
- For manually added movies, data such as title, description, rating, and poster URL will be stored in the application state.

CHAPTER: - 3
SYSTEM PLANNING

3.1 System Development Model:

The **System Development Model** outlines the approach used for developing the Movie Manager Application. For this project, we have chosen the **Agile Development Model** due to its iterative, user-centric approach and its ability to adapt to changing requirements throughout the development process.

3.1.1 Agile Development Model

Agile methodology is an iterative and incremental approach to software development, focusing on flexibility, collaboration, and delivering functional portions of the application at regular intervals. It is well-suited for projects like the Movie Manager Application, which requires continuous user feedback and frequent updates.

Key Characteristics of the Agile Model:

- **Iterations (Sprints):** The development is divided into small, time-boxed iterations (also called sprints) that usually last 1-2 weeks. Each sprint focuses on delivering specific features.
- **Frequent Releases:** After each sprint, a working version of the application is released with the implemented features.
- **Customer Feedback:** After each release, the development team gathers feedback from users and stakeholders to refine and improve the product.
- **Flexibility and Adaptation:** The Agile model allows for changes in project requirements even late in the development process, ensuring that the final product better meets user needs.

3.1.2 Phases in the Agile Development Model

The development process follows several phases, each contributing to the iterative process of refining and building the Movie Manager Application. These phases include:

1. **Planning Phase:**
   - Requirements are gathered from stakeholders (such as users and developers) to create a product backlog.
   - The features are prioritized, and user stories are defined for each sprint.
2. **Design and Development Phase:**
   - The initial iteration starts with the design of the Movie Manager Application's basic structure, including the user interface, state management, and core functionalities (movie search, adding, and deleting movies).
   - Development begins by implementing the most critical features first, such as fetching movie data from the OMDB API and managing the movie list in the application state.
3. **Testing Phase:**
   - Each iteration includes testing of the features developed during the sprint. Unit testing and integration testing are performed to ensure the correct functionality of each feature.
   - Continuous integration tools are used to ensure that the codebase remains stable after each change.
4. **Feedback and Refinement Phase:**
   - After each sprint, feedback is gathered from stakeholders, and the application is refined based on the feedback. This ensures the features align with user needs and expectations.
   - New features, improvements, and bug fixes are incorporated into future sprints.
5. **Release Phase:**
   - At the end of each sprint, a working version of the application is released. This version

includes all the features implemented during the sprint and is made available to users for evaluation.

6. **Deployment and Maintenance Phase:**
   - After the final iteration, the Movie Manager Application is deployed for public use. The deployment is followed by ongoing maintenance to fix bugs, enhance features, and address new requirements as they arise.

3.1.3 Why Agile for the Movie Manager Application?

- **User-Centric:** Agile development emphasizes collaboration with users throughout the project. This allows the development team to continuously gather user feedback and adjust the application to meet evolving needs.
- **Quick Iterations:** The iterative approach allows for rapid development and frequent releases, enabling users to experience new features sooner.
- **Flexible Design:** The flexibility of Agile ensures that the application can adapt to new requirements as they emerge, such as adding user authentication or integrating a backend for storing movie lists in future sprints.
- **Risk Reduction:** By testing and releasing features frequently, the Agile model minimizes the risk of major failure, as issues can be identified and resolved early in the development process.

3.1.4 Agile Artifacts

- **Product Backlog:** A prioritized list of all features, enhancements, and tasks needed for the Movie Manager Application. This backlog is continuously updated based on user feedback and project progress.
- **Sprint Backlog:** A subset of the product backlog selected for a specific sprint. It includes the features and tasks that will be developed during that sprint.
- **Increment:** The working version of the application developed during each sprint. The increment represents all the features and improvements completed in that sprint, ready for release or further testing.

3.2 System Architecture and Design:

In this section, we outline the high-level architecture and design principles followed for the Movie Manager Application.
System Architecture:
The application follows a **client-server architecture**. The frontend of the Movie Manager Application is built using **React.js** (JavaScript), while the backend (for future expansion) could be built using Node.js or another server-side technology. For this initial version, no backend is implemented, and the movie data is fetched directly from the OMDB API.
The system follows the **MVC (Model-View-Controller)** design pattern:

- **Model:** Handles the data logic and structure, such as storing movie details.
- **View:** The user interface (UI), built with React, displays the data to the user.
- **Controller:** The logic that manages the interaction between the view and the model. In this case, React components handle user actions, such as adding or deleting movies.
-

Design Considerations:
- **UI/UX Design:** The user interface is designed to be simple, intuitive, and responsive. The application must work seamlessly on both desktop and mobile devices.
- **State Management:** The system uses **React's useState** hook to manage the movie data locally. This enables users to add, delete, and modify the movie list without needing a

backend or database.

- **Performance:** The system should be optimized for performance, ensuring that the movie list is loaded quickly and the application remains responsive even with multiple movies in the list.
- **Security:** As no sensitive user data is handled in this version of the application, security considerations focus on ensuring that the application remains stable and free from security vulnerabilities.

3.3 Data Flow:

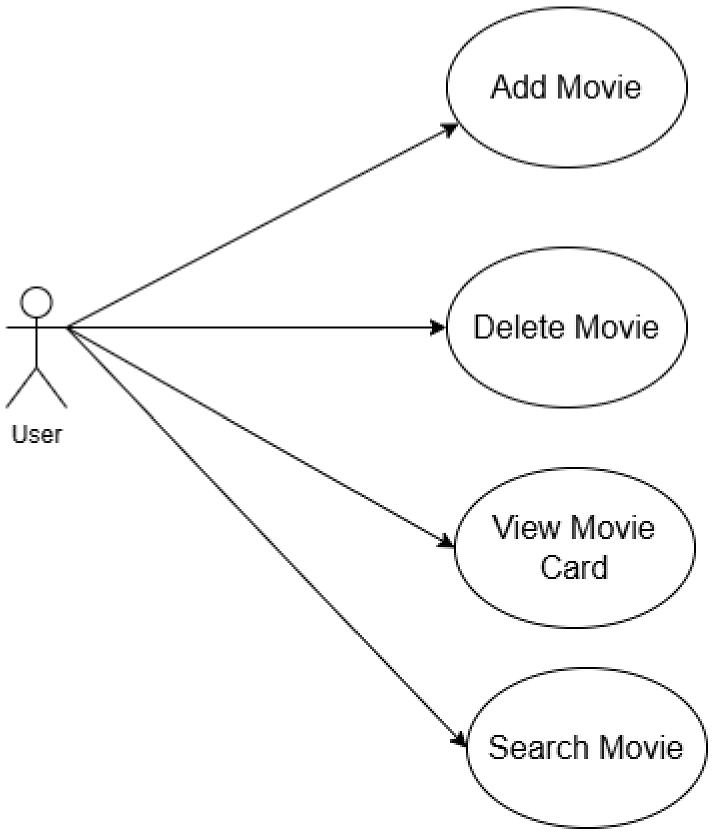
Data Flow Diagram (DFD):

The **Data Flow Diagram (DFD)** outlines how data moves through the system and interacts with various components. In this system, data primarily flows between the frontend (React.js) and the OMDB API.

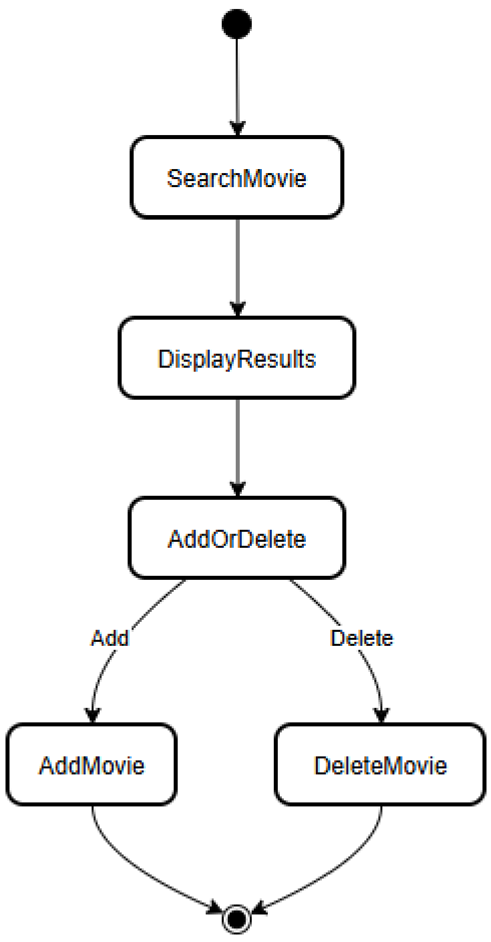
1. **User Actions:**
   - Users can search for movies, view details, add their own movies, and delete movies from the list.
2. **OMDB API Interaction:**
   - When a user searches for a movie, the system makes an API request to the OMDB API to fetch movie data.
   - The movie data is then displayed on the frontend, including title, year, description, rating, and poster image.
3. **State Management:**
   - Data is temporarily stored in the application state using React's useState hook.
   - When a user adds a new movie, the movie is stored in the state, and the UI is updated accordingly.
   - When a user deletes a movie, the state is updated to remove the movie from the list.

CHAPTER: - 4
SYSTEM DESIGN

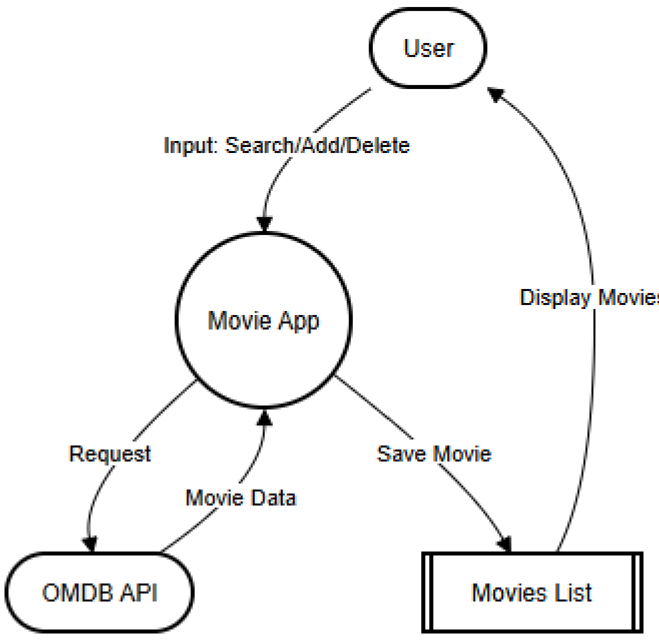
4.1 UseCase Diagram



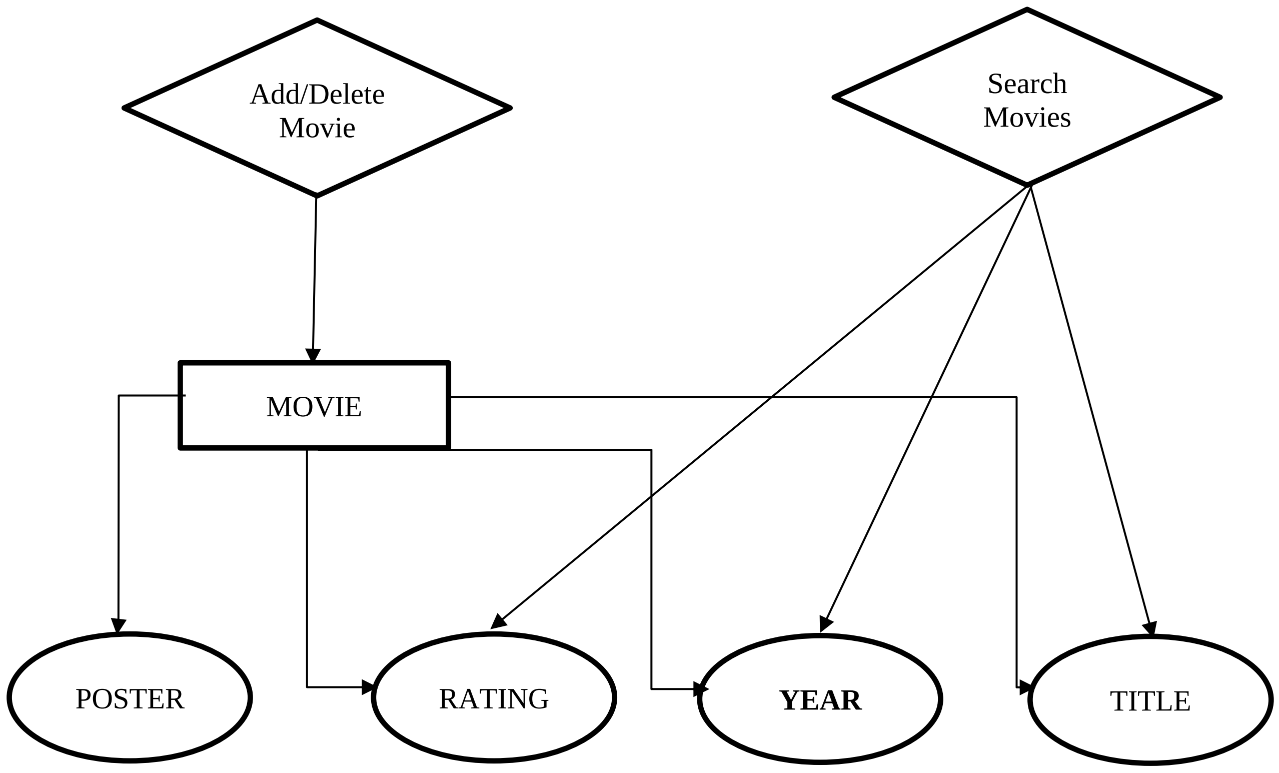
4.2 Activity Diagram



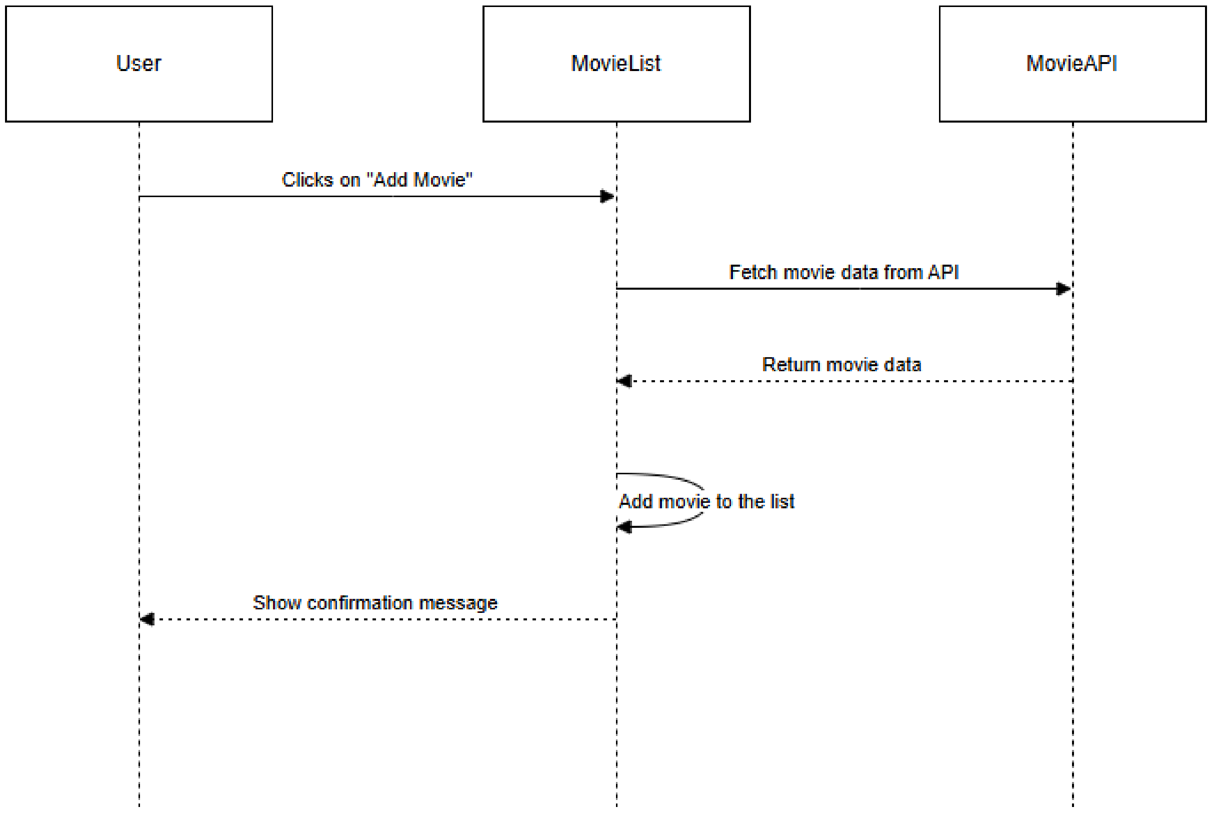
4.3 DataFlow Daigram



4.4 ER(Entity Relation) Diagram



4.5 Sequence Diagram



CHAPTER: - 5
IMPLEMENTATION

1.1 Hardware and Software Requirement

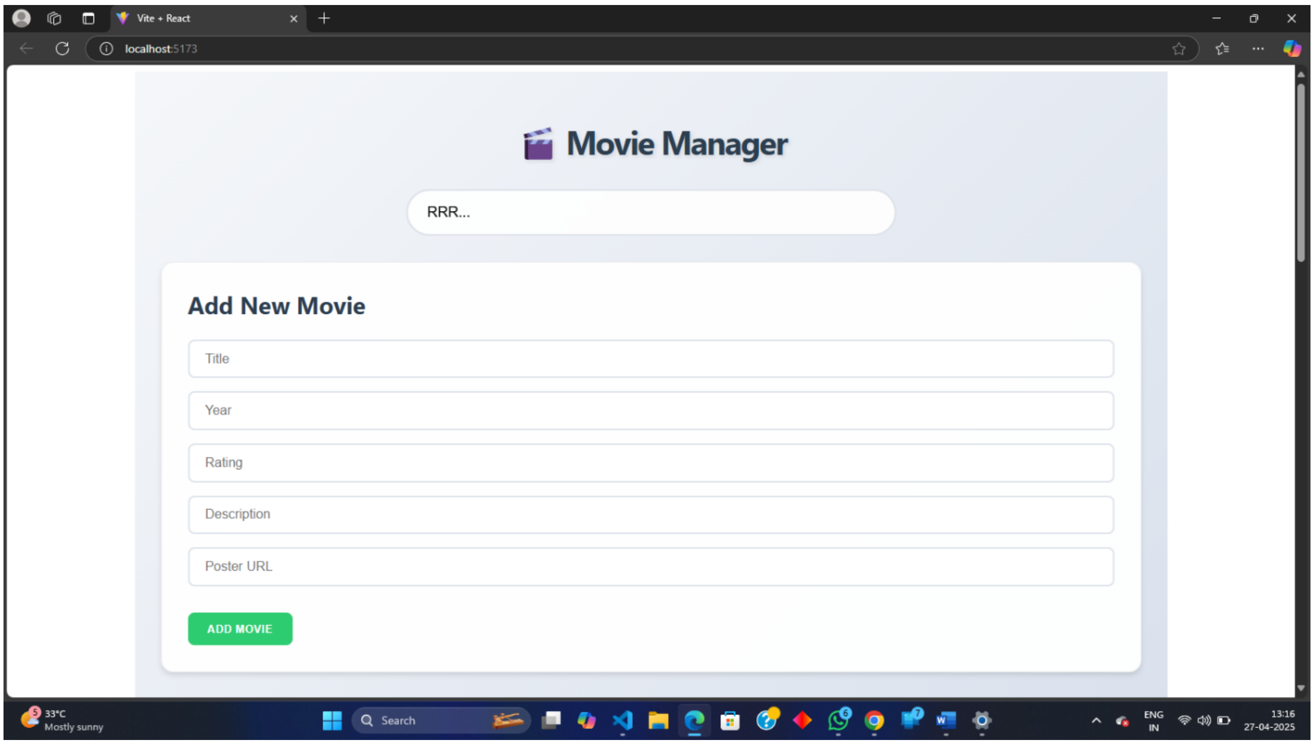
Hardware Requirements:

1. Processor: Intel i3 or higher, 2 cores
2. RAM: 4GB minimum, 8GB recommended
3. Storage: 10GB free space, SSD recommended
4. Display: 1366x768 resolution minimum, Full HD preferred
5. Internet Connection: Required for API and cloud access

Software Requirements:

1. Operating System: Windows 10+, macOS Mojave+, Linux (Ubuntu 20.04+)
2. Web Browser: Google Chrome, Mozilla Firefox, Microsoft Edge
3. Text Editor/IDE: Visual Studio Code, Sublime Text, or Atom
4. Node.js & npm: Latest LTS version for React setup
5. Vite: For fast project setup
6. Version Control: Git for version control, GitHub or GitLab for repository hosting
7. API: OMDB API for movie data (requires API key)
8. React Framework: React.js, React Router (optional)
9. Other Dependencies: Tailwind CSS/Bootstrap, Axios/Fetch API, React Hooks, ESLint, Prettier

5.2 Snapshot:-



Vite + React

localhost:5173

Movie Manager

RRR...

Add New Movie

Title

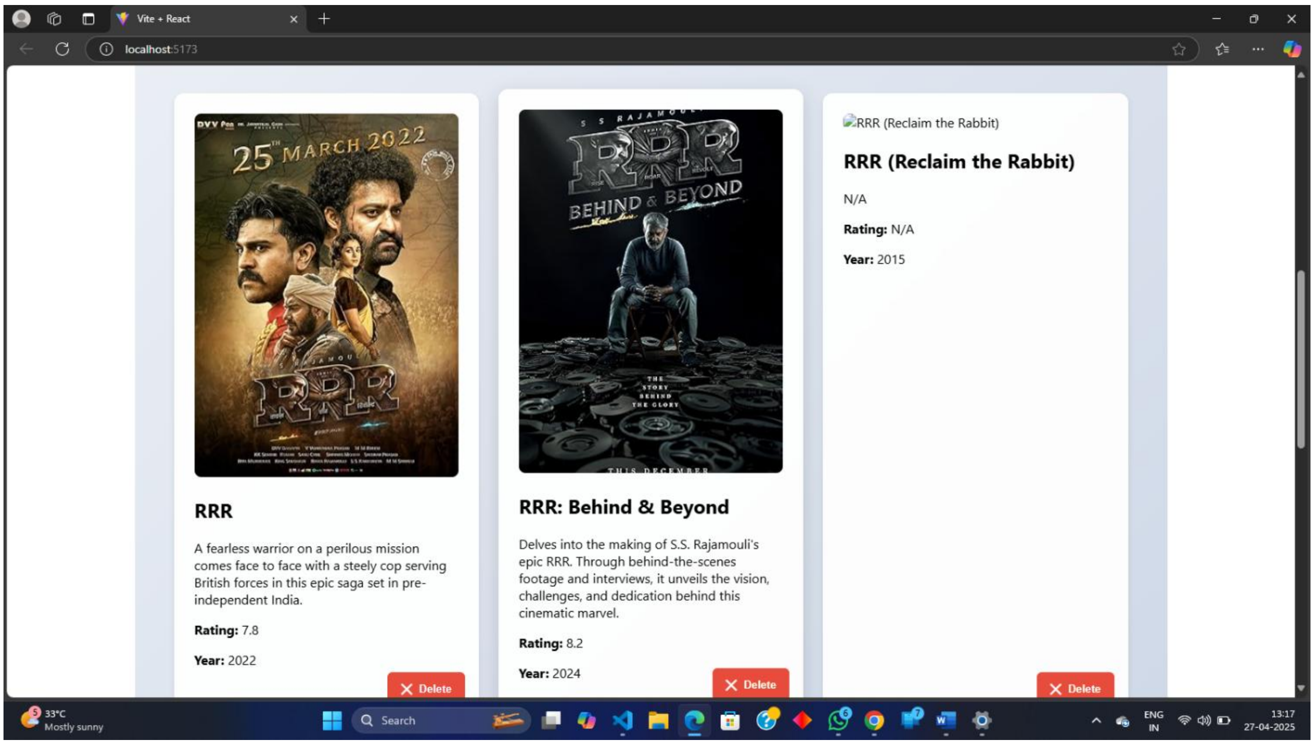
Year

Rating

Description

Poster URL

ADD MOVIE



Vite + React

localhost:5173

RRR

A fearless warrior on a perilous mission comes face to face with a steely cop serving British forces in this epic saga set in pre-independent India.

Rating: 7.8

Year: 2022

Delete

RRR: Behind & Beyond

Delves into the making of S.S. Rajamouli's epic RRR. Through behind-the-scenes footage and interviews, it unveils the vision, challenges, and dedication behind this cinematic marvel.

Rating: 8.2

Year: 2024

Delete

RRR (Reclaim the Rabbit)

RRR (Reclaim the Rabbit)

N/A

Rating: N/A

Year: 2015

Delete

CHAPTER: - 6
TEST

6.1Software Testing/Validation

Software testing and validation ensure that the application works as intended and meets the requirements. Below is the approach you can follow for testing your Movie List App:

 Unit Testing:

- Test individual components/functions (e.g., fetchMovies, handleAddMovie).
- Ensure correct behavior of small units.

 Integration Testing:

- Test if different parts of the app work together (e.g., adding a movie updates the list).
- Verify theme toggle interacts with components.

 Functional Testing:

- Test core features like search, add/delete movie.
- Ensure the app displays correct movie details.

 Usability Testing:

- Test if the app is user-friendly and easy to navigate.
- Check if UI is intuitive, and dark mode is functional.

 Performance Testing:

- Test load time and app performance with many movies.
- Ensure movie addition/deletion doesn’t slow down the app**.**

 Compatibility Testing:

- Test the app on different browsers (Chrome, Firefox, etc.).
- Ensure responsiveness across devices (mobile, desktop).

 Regression Testing:

- Re-test after changes to ensure no old features are broken.

 User Acceptance Testing (UAT):

- Get user feedback to ensure the app meets their needs before deployment.

6.2 Test Case

Test Case ID	Test Description	Test Steps	Expected Result	Status (Pass/Fail)
TC01	Test Search Functionality	1. Open the app. 2. Enter a movie title in the search bar. 3. Press Enter.	The movie list should display the searched movie.	Pass
TC02	Test Add Movie Functionality	1. Click on the "Add Movie" button. 2. Fill out the movie details. 3. Click on the "Add Movie" button.	The new movie should appear in the movie list.	Paas
TC03	Test Delete Movie Functionality	1. Open the app. 2. Select a movie to delete. 3. Click on the "Delete" button for that movie.	The selected movie should be removed from the movie list.	Pass
TC04	Test Dark Mode Toggle	1. Open the app. 2. Click on the "Toggle Dark Mode" button. 3. Observe the page's background color and text color.	The app should switch between light and dark mode.	Pass

CHAPTER: - 7
CONCLUSION AND FUTURE SCOPE

Conclusion

In conclusion, the Movie List project demonstrates an effective use of React to manage and display a list of movies, allowing users to search, add, and delete movies with real-time updates. The application also integrates features like dark mode, ensuring a user-friendly interface, and interacts with an external API (OMDB) to fetch movie data. Throughout the development process, key concepts such as state management, event handling, and conditional rendering have been implemented to build a responsive and functional app. The testing phase ensures the application functions properly by validating each feature, including search, add, delete, and dark mode toggling.

The project successfully meets its functional requirements, providing users with an intuitive movie management system. With thorough testing using both White Box and Black Box techniques, the system is robust, with minimal bugs. As it stands, the Movie List application serves as a strong foundation for understanding basic React concepts and APIs.

Future Scope

The Movie List project has significant potential for future enhancements and features. Some of the possible improvements and extensions include:

1. **User Authentication:**
    - Implement a login and registration system to allow users to save their personalized movie lists and preferences.
2. **Movie Details:**
    - Add a feature to display detailed information about each movie, including a plot summary, director, cast, etc., when a user clicks on a movie.
3. **Improved UI/UX:**
    - Enhance the UI with animations and transitions to make the application more engaging and user-friendly.
    - Consider adding responsive design features to ensure the app works seamlessly across mobile devices, tablets, and desktops.
4. **Advanced Search and Filtering:**
    - Implement advanced search filters such as sorting by genre, rating, year, and more.
    - Add pagination to manage large lists of movies or to provide infinite scrolling for a better user experience.
5. **Movie Ratings and Reviews:**

- Allow users to rate movies and add reviews, creating a more interactive and social experience.

6. **Offline Storage:**

   - Implement offline storage using localStorage or IndexedDB, so users can access their movie list even when not connected to the internet.

7. **Performance Optimization:**

   - Focus on performance improvements by lazy loading movie data or optimizing the API calls to reduce loading time.

8. **Integration with Other APIs:**

   - Integrate additional APIs (like TMDb or IMDb) to provide more comprehensive movie data and features.