An Industrial Oriented Mini Project Report on

REACT MOVIE APP

Submitted in Partial fulfillment of requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

AKULA SAI SAKETH	**20BD1A0525**
MEDISETTY BADRINATH	**20BD1A052Y**
POGULA CHANDRA KANTH	**20BD1A0538**
PUTTALA PRANAY TEJA	**20BD1A053A**

Under the guidance of

Ms.Sirisha K L S
Assistant Professor, Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH.

Narayanaguda, Hyderabad, Telangana-29

2023-24

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH.
Narayanaguda, Hyderabad, Telangana-29

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this is a bonafide record of the project report titled **"React Movie App"** which is being presented as the Industrial Oriented Mini Project report by

AKULA SAI SAKETH **20BD1A0525**

In partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering affiliated to the Jawaharlal Nehru Technological University Hyderabad, Hyderabad

Faculty Supervisor
(Ms.Sirisha K L S)

Head of Department
(Mr. P. Upender)

Submitted for Viva Voice Examination held on ______________________

External Examiner

Vision & Mission of KMIT

Vision of KMIT

- To be the fountainhead in producing highly skilled, globally competent engineers.
- Producing quality graduates trained in the latest software technologies and related tools and striving to make India a world leader in software productsand services.

Mission of KMIT

- To provide a learning environment that inculcates problem solving skills,professional, ethical responsibilities, lifelong learning through multi modal platforms and prepares students to become successful professionals.
- To establish an industry institute Interaction to make students ready for the industry.
- To provide exposure to students on the latest hardware and software tools.
- To promote research-based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises.
- To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effectivesolutions.
- To support the faculty to accelerate their learning curve to deliver excellent service to students.

Vision & Mission of CSE

Vision of the CSE

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

Mission of the CSE

- To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.
- To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.
- To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.
- To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.
- To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefit the society-at-large.
- To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities

PROGRAM OUTCOMES (POs)

PO1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution ofcomplex engineering problems.

PO2. Problem Analysis: Identify formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3. Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct Investigations of Complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern Tool Usage: Create select, and, apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The Engineer and Society: Apply reasoning informed by contextual knowledge to societal, health, safety. Legal und cultural issues and the consequent responsibilitiesrelevant to professional engineering practice.

PO7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-Long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Information Technology solutions for social upliftments.

PSO2: Shall have expertise on the evolving technologies like Python, MachineLearning, Deep learning, IOT, Data Science, Full stack development, Social Networks, Cyber Security, Mobile Apps, CRM, ERP, Big Data, etc.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields orwill be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in theirprofession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development byrapidly adapting to the changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit highlevels of professionalism and ethical responsibility.

PROJECT OUTCOMES

P1: Users can easily navigate through the application with a well-designed and responsive user interface.

P2: The React movie app delivers an immersive experience for users, utilizing React components to create engaging and dynamic UI elements. Users can seamlessly explore trending and upcoming movies, enhancing their overall movie-watching experience.

P3: The application adapts seamlessly to various devices, addressing environmental and sustainability concerns, and ensuring an equally functional and aesthetic experience across different screens.

P4: The project is well-documented, making it easy for developers to understand the codebase, structure, and functionality.

P5. : The React app includes an interactive video player, allowing users to play selected movies effortlessly. React app ability to handle interactive components ensures a smooth and enjoyable movie playback experience for users.

MAPPING PROJECT OUTCOMES WITH PROGRAM OUTCOMES

PO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
P1		H	L			H		H	M	M	L	L
P2	M	M	H		L	H	L	H	M	M	M	L
P3	H	H	H		H	H	L	H	M	M	M	L
P4	H	H	H		M	H		H	M	M	M	L
P5	M	M	M	L	M	H		H	H	H	H	H

L – LOW M –MEDIUM H– HIGH

PROJECT OUTCOMES MAPPING WITH PROGRAM SPECIFIC OUTCOMES

PSO	PSO1	PSO2
P1	H	M
P2	H	M
P3	H	H
P4	M	H
P5	H	M

PROJECT OUTCOMES MAPPING WITH PROGRAM EDUCATIONAL OBJECTIVES

PEO	PEO1	PEO2	PEO3	PE04
P1	H	H	M	
P2	M	M	M	
P3	H		M	
P4	H	H		L
P5	H		M	

DECLARATION

We hereby declare that the results embodied in the dissertation entitled "React Movie App" has been carried out by us together during the academic year 2023-24 as a partial fulfillment of the award of the B.Tech degree in Computer Science and Engineering from JNTUH. We have not submitted this report to any other university or organization for the award of any other degree.

Student Name	Roll no.
AKULA SAI SAKETH	20BD1A0529
MEDISETTY BADRINATH	20BD1A052Y
POGULA CHANDRA KANTH	20BD1A0538
PUTTALA PRANAY TEJA	20BD1A053A

ACKNOWLEDGEMENT

We take this opportunity to thank all the people who have rendered their full support to our project work. We render our thanks to **Dr. B L Malleswari**, Principal who encouraged us to do the Project.

We are grateful to **Mr. Neil Gogte**, Founder & Director, **Mr. S. Nitin,** Director for facilitating all the amenities required for carrying out this project.

We express our sincere gratitude to **Ms. Deepa Ganu**, Director Academic for providing an excellent environment in the college.

We are also thankful to **Mr. P Upender**, Head of the Department for providing us with time to make this project a success within the given schedule.

We are also thankful to our Faculty Supervisor **Ms. Sirisha K L S** , for her/his valuable guidance and encouragement given to us throughout the project work.

We would like to thank the entire CSE Department faculty, who helped us directly andindirectly in the completion of the project.

We sincerely thank our friends and family for their constant motivation during the project work.

Student Name	**Roll no.**
AKULA SAI SAKETH	**20BD1A0525**
MEDISETTY BADRINATH	**20BD1A052Y**
POGULA CHANDRA KANTH	**20BD1A0538**
PUTTALA PRANAY TEJA	**20BD1A053A**

ABSTRACT

The React movie app is a sophisticated web application designed to provide users with an intuitive platform for browsing, searching, and bookmarking their favorite movies. Leveraging ReactJS, TMDB API, Firebase Google Authentication, and Framer Motion, the application offers seamless functionality and an engaging user experience. The app enables users to search for movies by title, sort them by genre, access trending and upcoming movie lists, and view detailed information about each movie. Furthermore, users can securely sign in using their Google accounts, enhancing the personalization and security of the platform. This documentation comprehensively outlines the development process, technical architecture, feature implementations, and testing methodologies employed in creating the React movie app.

Additionally, it discusses potential future enhancements and troubleshooting guidelines to facilitate a deeper understanding of the project's intricacies and foster continued development and maintenance. This project serves as a testament to the integration of cutting-edge technologies to build an interactive and visually appealing movie-centric web application.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER – 1

1. INTRODUCTION

The React movie app is an innovative and user-friendly web application designed to cater to the entertainment needs of movie enthusiasts. With a focus on providing an immersive movie browsing experience, the application incorporates a range of features that enable users to explore, search, and interact with a vast collection of movies. Leveraging the power of ReactJS, the app ensures a dynamic and responsive user interface, enhancing the overall accessibility and engagement for movie enthusiasts of all kinds.

Integrating the robust TMDB (The Movie Database) API, the application offers users an extensive repository of movies, including popular titles, trending releases, and upcoming features. Users can effortlessly navigate through a rich library of movies, filter them by genre, and access comprehensive details about each film, including cast information, release dates, and user ratings.

To ensure a personalized and secure user experience, the React movie app implements Firebase Google Authentication, allowing users to conveniently sign in using their Google accounts. This feature not only streamlines the login process but also enhances data security and user interaction within the application.

Moreover, the application incorporates the dynamic animation capabilities of Framer Motion, contributing to a visually appealing and engaging user interface. This integration adds a touch of fluidity and interactivity, enriching the overall user experience and making the process of browsing and exploring movies more captivating and enjoyable.

This documentation provides a comprehensive overview of the development process, technical architecture, key features, testing methodologies, and potential future enhancements of the React movie app. By delving into the intricacies of the application's design and functionality, this documentation aims to offer insights into the implementation of modern technologies and best practices within the realm of web-based movie browsing platforms.

1.1 Purpose of the project

The purpose of the React movie app project is to provide a comprehensive and user-friendly platform for movie enthusiasts to explore, discover, and engage with a vast array of cinematic content. The project aims to simplify the movie-watching experience by leveraging the power of modern web technologies, including ReactJS, to create an intuitive and visually appealing user interface.

At its core, the project seeks to address the challenges users face when searching for movies, offering a solution that allows them to seamlessly navigate through an extensive database. Through the integration of the TMDB API, the app provides up-to-date information about movies, including details such as titles, genres, release dates, and overviews.

In essence, the purpose of the React movie app is to foster a sense of community among movie enthusiasts, providing a central hub where they can not only discover and bookmark their favorite movies but also share their passion with others. By creating a user-friendly and feature-rich platform, the project endeavors to enhance the overall enjoyment of the movie-watching journey for users of varying interests and preferences.

Background & Motivation:

The concept of the React movie app stemmed from a collective passion for movies and a recognition of the need for a seamless and interactive platform that could cater to the diverse preferences of movie enthusiasts. With the increasing accessibility of digital content and the growing demand for personalized entertainment experiences, there arose a compelling opportunity to create a user-centric application that could simplify the process of discovering and exploring a vast array of films.
In an era marked by the rapid digitization of media consumption, the team behind the React movie app recognized the significance of developing a technologically advanced solution that could leverage the power of contemporary web development frameworks. The selection of ReactJS as the primary framework was driven by its capacity to facilitate the creation of dynamic and responsive user interfaces, thereby ensuring an engaging and intuitive movie browsing experience for users of varying technological proficiency.

1.2 Problem with existing systems

Existing movie-related platforms often face several challenges that hinder the user experience. Many platforms lack an intuitive and visually engaging interface, making movie discovery and exploration cumbersome for users. Limited personalization features and inadequate search functionalities contribute to ageneric experience, where users struggle to find movies tailored to their preferences. Authentication and bookmarking processes can be convoluted, diminishing the ease of use. Additionally, outdated information and a lack of real-time updates on trending and upcoming movies further detract from the overall user satisfaction. Addressing these issues forms the motivation behind developing the React movie app, aiming to overcome the shortcomings of existing systems and provide an enhanced, user-centric movie-watching platform.

1.3 Proposed System

The proposed React movie app envisions a dynamic and user-centric platform that revolutionizes the movie-watching experience. Rooted in ReactJS and seamlessly integrated with the TMDB API, the system boasts a modern and visually appealing interface, addressing the limitations of existing platforms. The app empowers users to effortlessly search, explore, and discover movies, offering a comprehensive database enriched with real-time updates.

One of the key innovations lies in the implementation of Firebase Google Authentication, streamlining user login processes for heightened security and personalized features. Framer Motion introduces an element of sophistication, enhancing the user interface with fluid animations. The proposed system not only solves existing issues related to movie discovery but also prioritizes user engagement by allowing seamless bookmarking of favorite movies and providing a range of search and sorting options.

Moreover, the React movie app goes beyond conventional platforms by offering curated sections for trending and upcoming movies, ensuring users stay informed about the latest releases. The system's architecture is designed to foster a sense of community among movie enthusiasts, allowing them to share their passion and create a personalized movie-watching environment. In essence, the proposed system strives to set a new standard in the realm of movie-related platforms, combining cutting-edge technology with user-centric design for an unparalleled cinematic journey.

1.4 Scope of the project

Scope of the project:

• Create a vast and diverse movie database encompassing a wide array of genres, languages, and release years to cater to the varied preferences of global users.

• Develop an intuitive and visually appealing user interface that facilitates effortless navigation, enabling users to explore and discover movies seamlessly.

• Implement a sophisticated recommendation system based on user preferences and viewing history, offering personalized movie suggestions and tailored content to each user.

• Enable efficient sorting and filtering options to allow users to organize movies based on genres, release dates, popularity, and user ratings for an optimized browsing experience.

1.5 Architecture Diagram

"The architectural diagram for the React movie app project provides a visual representation of the system's structure and components. This diagram illustrates how different elements, including React components, API integrations, authentication processes, and external services, interact to deliver the intended functionality. By presenting a high-level overview of the project's architecture, this diagram serves as a valuable reference for developers, stakeholders, and anyone seeking insights into the system's design and flow. It showcases the seamless integration of technologies such as ReactJS, TMDB API, Firebase Authentication, and Framer Motion, offering a clear understanding of how these components contribute to the overall functionality and user experience of the application."

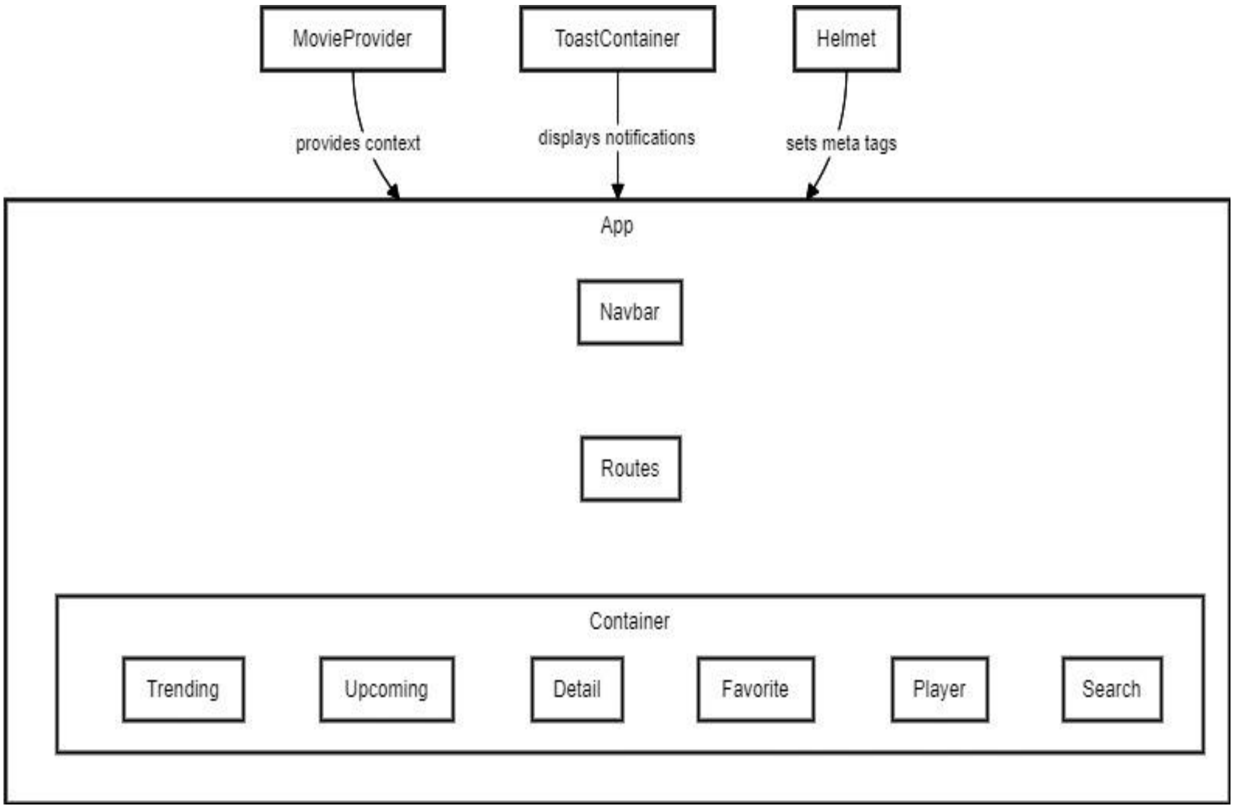


Fig 1.5.1 Architectural Diagram

CHAPTER – 2

2.LITERATURE SURVEY

The process of summarization of the previous research works on the particular field of research. The aim of every research is deriving a new solution or invention. There are mainly five steps in literature survey they are as follows

- Collect
- Study
- Analyse
- Identify
- Summarization

1)Environment Setup:

The project begins by detailing the setup of a React development environment. It may include instructions for installing Node.js, creating a new React application using tools like Create React App, and setting up the project structure.

2)API Integration:

A central aspect of the project would be fetching movie data from an API. This could involve utilizing JavaScript's fetch API or external libraries like Axios to retrieve movie information from online sources.

3) Component-Based Architecture:

The project would likely emphasize the fundamental concept of component-based architecture in React. It could cover the creation and organization of components, such as MovieList, MovieCard, SearchBar, and others.

4)Dynamic Rendering:

A crucial aspect of the project may involve dynamically rendering movie data fetched from the API. It might demonstrate how to iterate over the data and render movie cards with relevant information like title, poster, and description.

5)Search and Filtering Functionality:

The project will provide insights into implementing search and filtering features, allowing users to search for specific movies or filter results based on various criteria.

6)Responsive Design:

Given the importance of mobile accessibility, the project will discuss about responsive design principles to ensure the movie app looks and functions well across different devices and screen sizes.

7)React Router and Views:

The project uses React Router, a library for handling navigation and views within a React application. This could enable developers to create different views for displaying movie details, search results, and more.

8)Styling and User Interface Design:

The tutorial would likely address styling and user interface design considerations. It may offer guidance on using CSS or CSS-in-JS libraries to style components and enhance the app's visual appeal.

CHAPTER – 3

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 Introduction to SRS

Software Requirement Specification (SRS) is an essential document that serves as the starting point for software development activities. As systems grow more complex, it becomes evident that comprehending the goals of the entire system is not an easy task. This realization has led to the emergence of the requirement phase. which aims to address this challenge.

The software project is initiated by the needs and requirements of the client. The SRS plays a crucial role in translating the ideas and visions of the clients into a formal document that serves as the output of the requirement phase. It acts as a communication bridge between the clients and the development team, ensuring that the client's expectations and requirements are accurately captured and understood.

The SRS phase consists of two basic activities:

Problem/Requirement Analysis:

The problem/requirement analysis phase is a crucial step in the software development process. It involves a systematic approach to understanding the problem, goals, and constraints associated with the software project. This phase is often considered more nebulous and less structured compared to the requirement specification phase. Its main objective is to gain a deep understanding of the client's needs and expectations.

During the problem/requirement analysis phase, several key activities take place to gather relevant information and define the project's scope:

1. Problem Identification:

The first step is to identify and define the problem that the software system aims to solve. This involves analysing the current processes, identifying pain points, and understanding the challenges faced by the client. By comprehending the problem at hand, the development team can align their efforts towards finding an effective solution.

2. Goal Definition:

Once the problem is identified, the next step is to define clear and measurable goals for the software project. These goals serve as the guiding principles for the development process and help in determining the success criteria. Goals can be related to improving efficiency, enhancing user experience, increasing productivity, or achieving specific business objectives.

3. Requirement Elicitation:

This activity involves gathering requirements directly from the client, end- users, and other stakeholders. It may include interviews, surveys, workshops, or other techniques to understand their needs and expectations. The focus is on eliciting both functional and non-functional requirements to capture the complete picture of what the software system should achieve.

4. Requirement Analysis:

Once the requirements are gathered, they are analyzed to identify dependencies, conflicts, and any missing information. The development team reviews the requirements in detail, seeks clarifications from the client if needed, and ensures that they are feasible and aligned with the project's goals. This analysis helps in refining and prioritizing the requirements.

5. Constraint Identification:

In addition to understanding the problem and goals, it is essential to identify any constraints or limitations that may impact the software development process. Constraints can be related to budget, time, technology, regulations, or existing infrastructure. Identifying these constraints upfront allows for better planning and decision-making during the subsequent phases.

6. Scope Definition:

Based on the problem analysis, goal definition, requirement elicitation, and constraint identification, the scope of the software project is defined. The scope outlines the boundaries of the system, specifying what functionalities will be included and what will be excluded. It helps in managing expectations and ensures that the project remains focused and achievable within the given constraints.

7. Documentation:

Throughout the problem/requirement analysis phase, documentation plays a crucial role. The findings, requirements, goals, and constraints are documented in a structured manner to ensure clarity and shared understanding among all stakeholders. This documentation serves as a reference for the development team and helps in the subsequent phases of the project.

Requirement Specification:

The problem/requirement analysis phase is a crucial step in the software development process. It involves a systematic approach to understanding the problem, goals, and constraints associated with the software project. This phase is often considered more nebulous and less structured compared to the requirement specification phase. Its main objective is to gain a deep understanding of the client's needs and expectations.

During the problem/requirement analysis phase, several key activities take place to gather relevant information and define the project's scope:

1. Problem Identification: The first step is to identify and define the problem that the software system aims to solve. This involves analysing the current processes, identifying pain points, and understanding the challenges faced by the client. By comprehending the problem at hand, the development team can align their efforts towards finding an effective solution.

2. Goal Definition: Once the problem is identified, the next step is to define clear and measurable goals for the software project. These goals serve as the guiding principles for the development process and help in determining the success criteria. Goals can be related to improving efficiency, enhancing user experience, increasing productivity, or achieving specific business objectives.

3. Requirement Elicitation: This activity involves gathering requirements directly from the client, end-users, and other stakeholders. It may include interviews, surveys, workshops, or other techniques to understand their needs and expectations. The focus is on eliciting both functional and non-functional requirements to capture the complete picture of what the software system should achieve.

4. Requirement Analysis: Once the requirements are gathered, they are analyzed to identify dependencies, conflicts, and any missing information. The development team reviews the requirements in detail, seeks clarifications from the client if needed, and ensures that they are feasible and aligned with the project's goals. This analysis helps in refining and prioritizing the requirements.

5. Constraint Identification: In addition to understanding the problem and goals, it is essential to identify any constraints or limitations that may impact the software development process. Constraints can be related to budget, time, technology. regulations, or existing infrastructure. Identifying these constraints upfront allows for better planning and decision-making during the subsequent phases.

6. Scope Definition: Based on the problem analysis, goal definition, requirement elicitation, and constraint identification, the scope of the software project is defined. The scope outlines the boundaries of the system, specifying what functionalities will be included and what will be excluded. It helps in managing expectations and ensures that the project remains focused and achievable within the given constraints.

7. Documentation: Throughout the problem/requirement analysis phase, documentation plays a crucial role. The findings, requirements, goals, and constraints are documented in a structured manner to ensure clarity and shared understanding among all stakeholders. This documentation serves as a reference for the development team and helps in the subsequent phases of the project.

The requirement specification phase is a critical step in the software development process, following the problem/requirement analysis phase. In this phase, the focus is on specifying and documenting the requirements that have been identified during the analysis phase. The goal of this phase is to create a comprehensive and well-defined Software Requirement Specification (SRS) document that serves as a reference for the development team.

During the requirement specification phase, the following activities take place:

1. Specification Representation: The requirements identified in the analysis phase need to be represented in a clear and structured manner. This involves choosing appropriate representation techniques, such as diagrams, flowcharts, use cases, or user stories, to effectively communicate the requirements. The chosen representations should capture the essential functionality and behavior of the software system.

2. Specification Languages and Tools: Depending on the complexity and nature of the project, specific specification languages and tools may be utilized to document the requirements. These languages and tools provide a standardized way of expressing the requirements, ensuring clarity and consistency. Examples of such languages include Unified Modeling Language (UML), Business Process Model and Notation (BPMN), or specific domain-specific languages (DSLS).

3. Requirement Documentation: The identified requirements are documented in detail, specifying their functional and non-functional aspects. Functional requirements describe what the software system should do, while non-functional requirements outline the quality attributes, performance expectations, security considerations, and other constraints. Each requirement should be clearly defined, unambiguous, and verifiable to ensure that it can be effectively implemented and tested.

4. Requirement Validation: During the requirement specification phase, it is crucial to validate the documented requirements. This involves reviewing the requirements with the client and other stakeholders to ensure their accuracy, completeness, and alignment with the project goals. Any inconsistencies or gaps in the requirements are identified and addressed to avoid misunderstandings and future rework.

5. SRS Document Production: The culmination of the requirement specification phase is the production of the validated SRS document. This document serves as a formal record of the agreed-upon requirements and provides a comprehensive overview of the software system to be developed. The SRS document acts as a contract between the client and the development team, guiding the subsequent development, testing, and implementation activities.

3.2 Role of SRS

The role of the Software Requirement Specification (SRS) is crucial in bridging the communication gap between clients and developers during the software development process. The SRS serves as a formal document that accurately specifies the needs and requirements of the client and users. It forms the foundation and basis for the entire software development lifecycle.

1. Communication Bridge: The SRS acts as a communication bridge between clients and developers. It serves as a common reference point that ensures a shared understanding of the software project. By documenting the requirements in a clear and structured manner, the SRS helps to minimize misunderstandings and ambiguities, facilitating effective communication between all stakeholders.

2. Accurate Requirement Specification: The SRS document captures the client's and users' needs, expectations, and goals for the software system. It provides a detailed and comprehensive description of the desired functionalities, features, and constraints. By accurately specifying these requirements, the SRS helps to align the development team's efforts with the client's vision, ensuring that the final product meets their expectations.

3. Basis for Development: The SRS serves as the foundation for the software development process. It provides developers with a clear roadmap of what needs to be built, guiding them in the design, coding, testing, and implementation phases. The SRS ensures that the development team focuses their efforts on building the software system according to the specified requirements, minimizing the risk of scope creep and ensuring a more efficient development process.

4. Stakeholder Satisfaction: A good SRS aims to satisfy all the parties involved in the system. By accurately capturing the needs and expectations of clients and users, the SRS helps to ensure that the software system delivers the desired functionality, usability, and performance. It acts as a reference point for evaluating the success of the project, enabling stakeholders to assess whether the developed software meets their requirements and provides value.

5. Contractual Agreement: The SRS document serves as a contractual agreement between the client and the development team. It outlines the agreed-upon requirements, scope, and deliverables of the project. The SRS document provides a basis for managing expectations, resolving disputes, and ensuring that both parties are aligned throughout the software development process.

6. Change Management: The SRS also plays a role in managing changes during the software development lifecycle. As requirements evolve or new insights emerge, the SRS document can be updated and revised to reflect the changes. This helps to maintain a clear record of the project's requirements and facilitates effective change management, ensuring that modifications are properly communicated, evaluated, and implemented.

3.3 Requirements Specification Document

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application. There are a set of guidelines to be followed while preparing the software requirement specification document. This includes the purpose, scope, functional and non-functional requirements, software and hardware requirements of the project. In addition to this, it also contains the information about environmental conditions required, safety and security requirements, software quality attributes of the project etc.

The purpose of SRS (Software Requirement Specification) document is to describe the external behaviour of the application developed or software. It defines the operations, performance and interfaces and quality assurance requirement of the application or software. The complete software requirements for the system are captured by the SRS.

3.4 Functional Requirements

For documenting the functional requirements, the set of functionalities supported by the system are to be specified. A function can be specified by identifying the state at which data is to be input to the system, its input data domain, the output domain, and the type of processing to be carried on the input data to obtain the output data. Functional requirements define specific behaviour or function of the application. Following are the functional requirements:

1. User Registration and Authentication:

Users must be able to register for an account using their Google credentials through Firebase Google Authentication. Authenticated users should have their own profiles with personalized settings and preferences.

2. Movie Search and Discovery:

Users should be able to search for movies by title, providing relevant search results. The platform should support filtering and sorting options, including genre-based categorization. Users should be able to access detailed information about each movie, including cast and crew.

3. Watchlist and Bookmarking:

Users must be able to create and manage their watchlists, allowing them to bookmark their favorite movies for future viewing. The watchlist should provide easy access to saved movies and enable users to mark movies as "watched" or "unwatched."

4.Trending and Upcoming Movies:

The application must display a section showcasing trending movies, keeping users informed about what's currently popular. An upcoming movies section should inform users about future releases, building anticipation for upcoming titles.

5.Movie Recommendations:

The platform could provide movie recommendations based on a user's watch history or preferences. Users should have the option to explore personalized movie suggestions.

6.User Interaction and Feedback:

The application must facilitate user interactions, such as liking, rating, and reviewing movies. Users should be able to leave comments and reviews for movies and engage in discussions with other users.

7.Language and Region Support:

The application should provide support for multiple languages and regions, allowing users to navigate and access movie information in their preferred language and region.

3.5 Non-Functional Requirements

Non-functional requirements are an important aspect of the Software Requirement Specification (SRS) document as they define the constraints and qualities that the system must possess. These requirements focus on aspects other than the specific behavior of the system. Let's expand on the non-functional requirements mentioned:

1. Performance:

- Response Time: The application should respond to user actions quickly, with minimal latency, ensuring that users can navigate and access information without delays.

- Scalability: The system should be able to handle a growing user base and increasing data loads while maintaining performance. It should scale horizontally and vertically as needed.

- Load Handling: The platform should be capable of handling high traffic loads, particularly during peak usage times, without performance degradation.

2. Security:

• Authentication and Authorization: The application must ensure secure user authentication and authorization, protecting user accounts and personal information.

• Secure APIs: Access to APIs and data sources should be protected through proper authentication and authorization mechanisms.

3. Usability:

• User Interface Design: The user interface should be intuitive, visually appealing, and user-friendly, ensuring that users can easily navigate and interact with the application.

• Accessibility: The platform must adhere to accessibility standards, making it accessible to users with disabilities, including support for screen readers and keyboard navigation.

4. Reliability:

• Availability: The application should be available and operational 24/7, with minimal downtime for maintenance and updates.

• Fault Tolerance: The system should be designed to handle errors and failures gracefully, ensuring that a single point of failure does not disrupt the entire application.

5. Scalability and Extensibility:

The application architecture should be designed in a modular and extensible manner, allowing for the addition of new features and integrations without significant code rewrites.

6. Compatibility:

The platform should be compatible with a wide range of web browsers and devices, ensuring a consistent and enjoyable experience for users across different platforms.

3.6 Performance Requirements

1. Response Time:

- The system should respond to user interactions, such as searches and clicks, within 1-2 seconds to maintain a responsive feel.

2. Page Load Time:

- The initial loading time of the application should be optimized to ensure quick access for users. Aim for a page load time of under 3 seconds.

3. Scalability:

- The application should handle a large number of concurrent users without significant degradation in performance. Scalability testing should be conducted to ensure responsiveness under varying user loads.

4. Search Speed:

- The search functionality should provide real-time results, and the search process should be optimized to deliver accurate and quick responses.

5. Image Loading:

- Images, especially movie posters and thumbnails, should load efficiently. Implement lazy loading or progressive loading techniques to enhance the speed of image rendering.

6. API Response Time:

- Requests to external APIs, such as the TMDB API, should be optimized to ensure quick response times. Asynchronous loading and caching mechanisms can be implemented to enhance API call efficiency.

3.7 Software Requirements

1. Development Tools:

• Code Editor: A code editor such as Visual Studio Code, Sublime Text, or JetBrains WebStorm for writing and editing code.

2. Front-End Frameworks and Libraries:

• ReactJS: The core front-end framework for building the user interface of the application.

• Framer Motion: A motion and animation library for creating engaging user interface animations and transitions.

3. Back-End Technologies:

• Node.js: A JavaScript runtime for running server-side code and handling server-side logic.

• Express.js: A web application framework for building the back-end server and APIs.

4. API Integration:

• TMDB API: Integration with The Movie Database (TMDB) API to access movie data, including movie details, images, and information.

5. Database Management Tools:

• Database management tools for interacting with and managing the project's database system.

6. Operating Systems:

• The development and deployment of the project can be carried out on various operating systems, including Windows, macOS, and Linux.

7. Web Browsers:

• Compatibility with multiple web browsers for testing and ensuring a consistent user experience.

3.8 Hardware Requirements

Processor	**Intel i3 or more**
RAM	4 GB or More
Hard disk	10GB or more
Monitor	15" CRT, or LCD monitor
Camera	With good resolution
Keyboard	Normal or Multimedia
Mouse	Compatible mouse

Table 3.8.1 Hardware Requirements

CHAPTER – 4

4. SYSTEM DESIGN

4.1 Introduction to UML

The Unified Modelling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic, semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagrams, which is as follows:

1. User Model View:
   This view represents the system from the users' perspective. The analysis representation describes a usage scenario from the end-users' perspective.

2. Structural Model View:
   In this model, the data and functionality are arrived from inside the system. This model view models the static structures.

3. Behavioral Model View:
   It represents the dynamic of behavioral as parts of the system, depicting he interactions of collection between various structural elements described in the user model and structural model view.

4. Implementation Model View:
   In this view, the structural and behavioral as parts of the system are represented as they are to be built.

5. Environmental Model View:
   In this view, the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

4.2 UML Diagrams

4.2.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. To clarify a bit in details, dynamic behavior means the behavior of the system when it is running/operating. So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So, use case diagrams are consisting of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So, to model the entire system numbers of use case diagrams are used. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analysed to gather its functionalities use cases are prepared and actors are identified. In brief, the purposes of use case diagrams can be as follows:

a. Used to gather requirements of a system.

b. Used to get an outside view of a system.

c. Identify external and internal factors influencing the system.

d. Show the interacting among the requirements are actors.

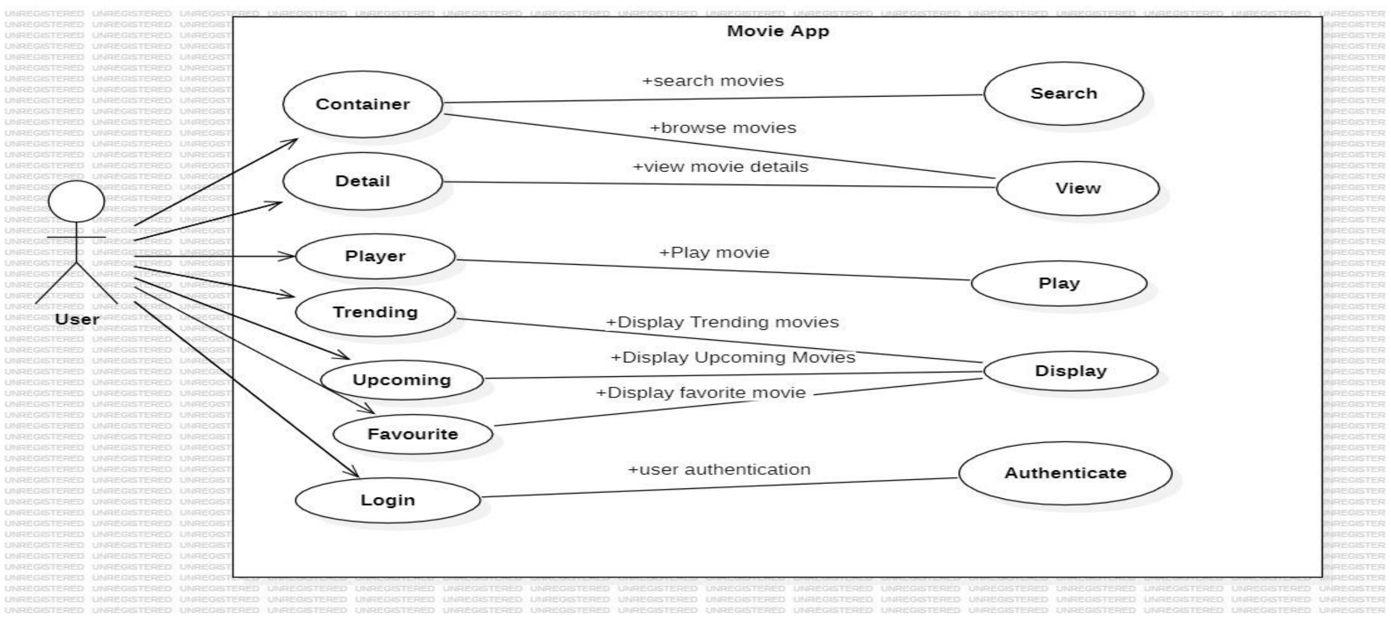


Figure 4.2.1.1 Use Case Diagram

4.2.2 Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modelling a new system. The aim of a sequence diagram is to define event sequences, which would have a desired outcome. The focus is more on the order in which messages occur than on the message. However, the majority of sequence diagrams will communicate what messages are sent and the order in which they tend to occur.

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes. Activation or Execution Occurrence Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin grey rectangle placed vertically on its lifeline.

Messages

Messages are arrows that represent communication between objects. Use half- arrowed lines to represent asynchronous messages.

Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.

Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.

Objects can be terminated early using an arrow labelled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets. Guards

When modelling object interactions, there will be times when a condition must be met for a message to be sent to an object. Guards are conditions that need to be used throughout UML diagrams to control flow.

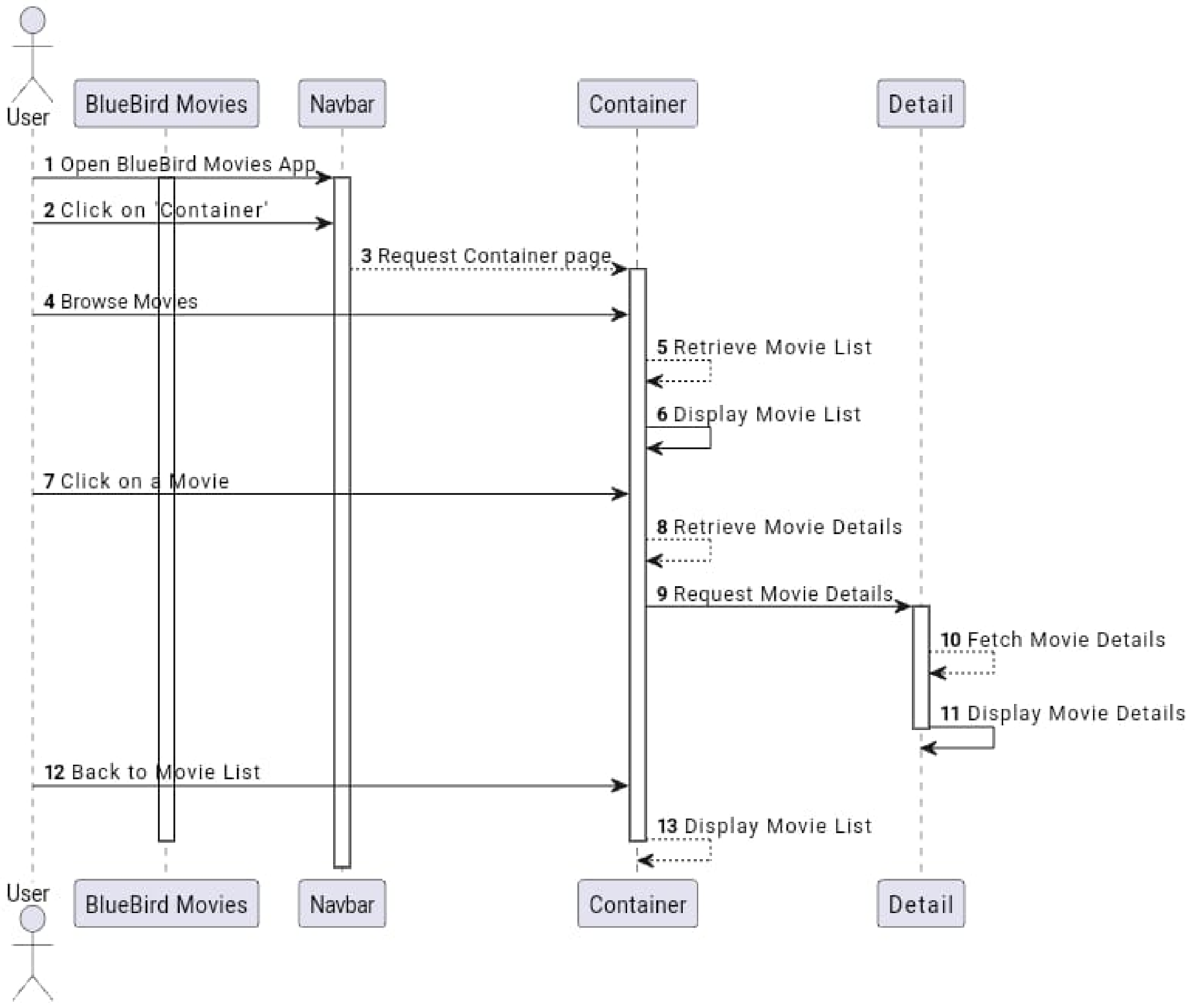


Figure 4.2.2.1 Sequence Diagram

4.2.3 State Chart Diagram

In UML (Unified Modeling Language), a state diagram is a type of behavioral diagram that is used to represent the dynamic behavior of an object or a system in response to various external and internal events. State diagrams are particularly useful for modeling the behavior of objects that can exist in multiple states and transition between these states based on specific conditions or events.

State diagrams are valuable for visualizing the behavior of complex systems, such as software applications, hardware devices, or business processes. They help in understanding how objects or systems respond to various events and conditions, making them an essential tool for modeling and designing software and systems with dynamic behavior.

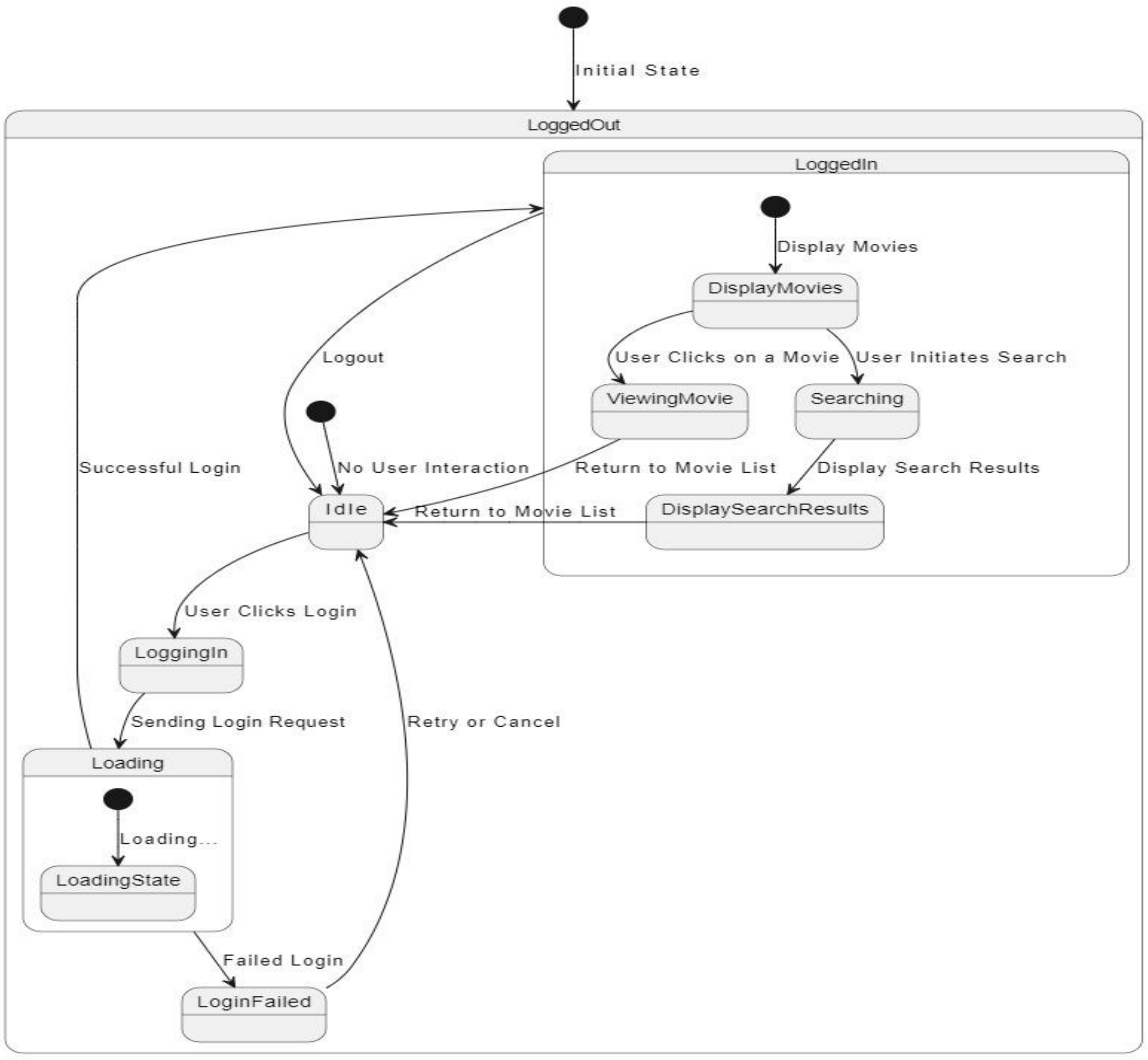


Figure 4.2.3.1 State Chart Diagram

4.2.4 Deployment Diagram

A deployment diagram in Unified Modeling Language (UML) is a type of diagram that visualizes the physical deployment of software components, hardware devices, and the connections between them in a system. It provides a clear view of how software and hardware elements are distributed and interact in a real-world environment.

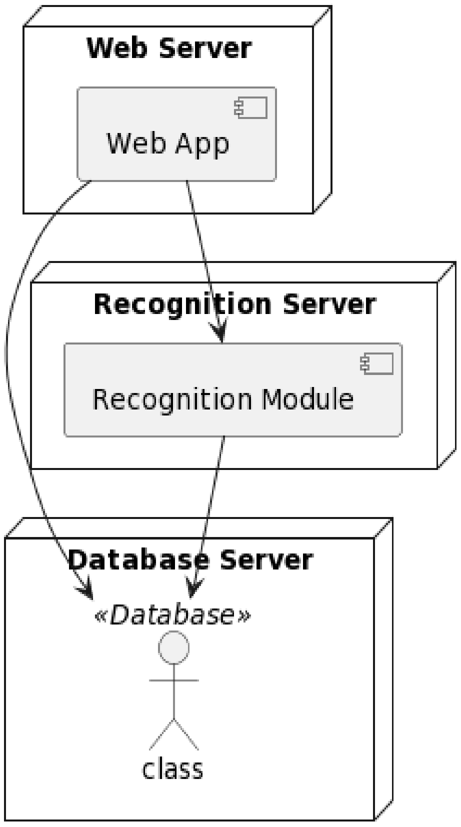
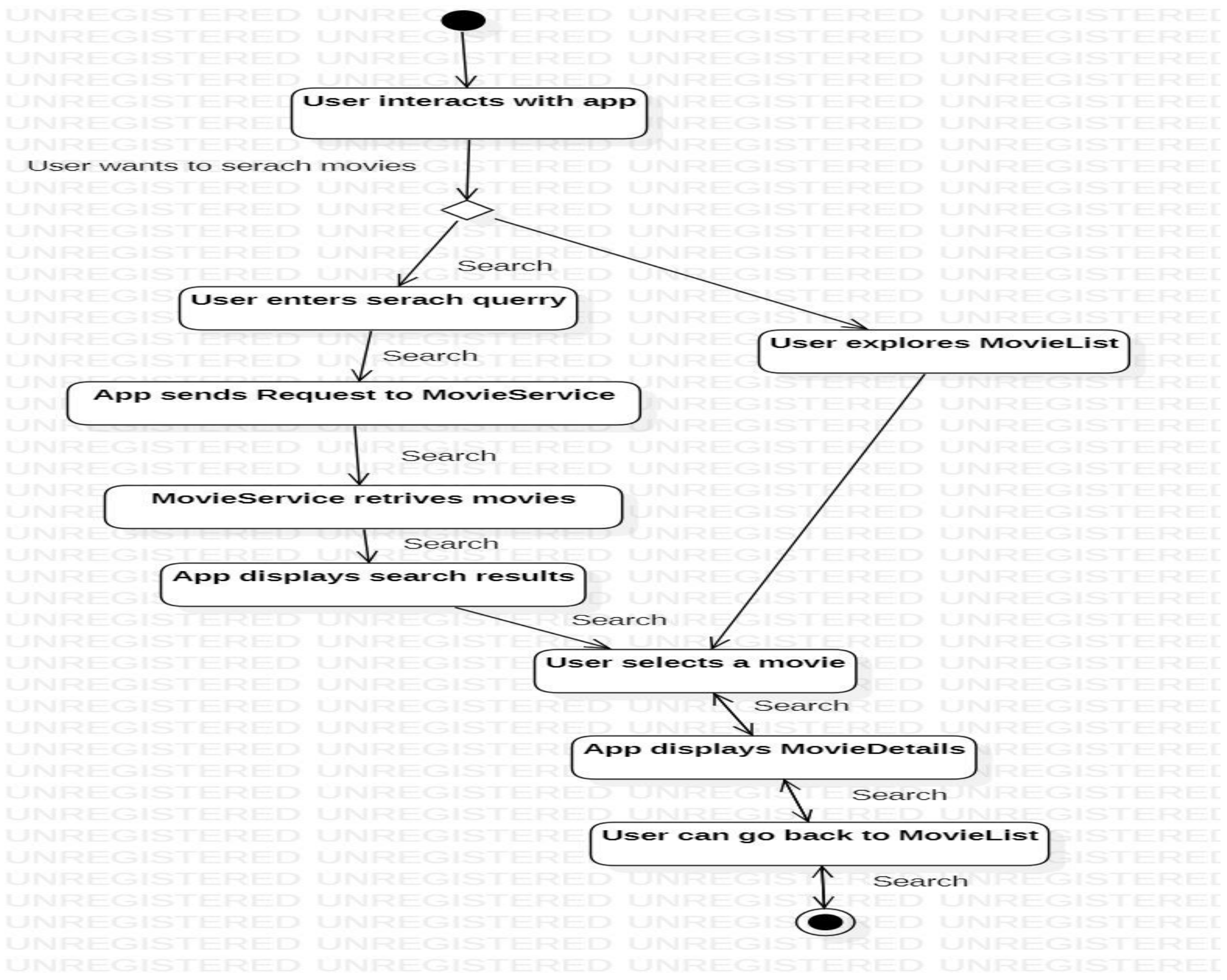


Figure 4.2.4.1 Deployment Diagram

4.2.5 Activity Diagram



User interacts with app
User wants to serach movies
Search
User enters serach querry
User explores MovieList
Search
App sends Request to MovieService
Search
MovieService retrives movies
Search
App displays search results
Search
User selects a movie
Search
App displays MovieDetails
Search
User can go back to MovieList
Search

Figure 4.2.5.1 Activity Diagram

Admin:

The admin is responsible to register new students, store images, train and start the system.

- The student is not required to do any manual work for attendance.
- The flow of the system starts with the images of the students that are stored in the database
- Then the recognition process starts.
- The camera captures the faces of the students and compares it with the images of the students in the database
- If the match is found then the student is marked as present by adding the details to a CSV file
- If there is no match then the flow returns to the recognition state.

4.2.6 Class Diagram

Class diagrams are the main building blocks of every object-oriented method. The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, so as the class diagram has appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in its context. It describes various kinds of objects and the static relationship in between them.

The main purpose to use class diagrams are:

1. This is the only UML which can appropriately depict various aspects of OOPs concept.

2.Proper design and analysis of application can be faster and efficient.

3.It is base for deployment and component diagram.

Each class is represented by a rectangle having a subdivision of three compartments name, attributes and operation

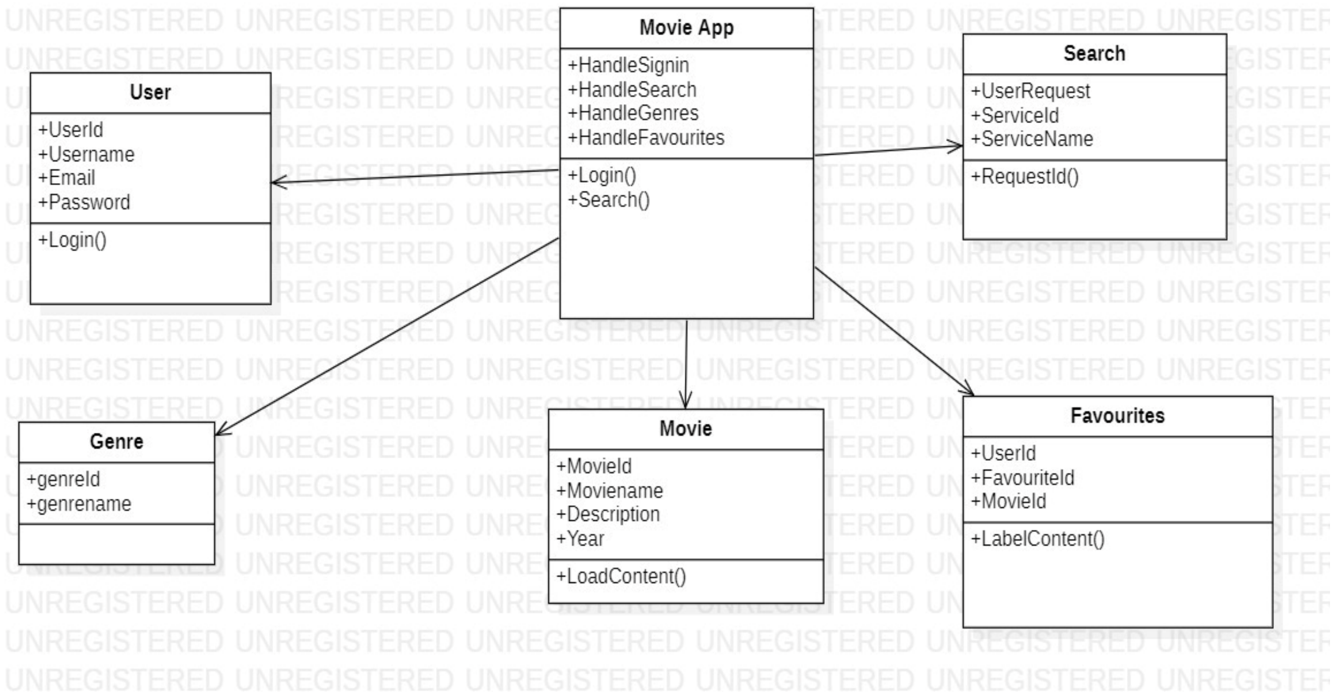


Figure 4.2.6.1 Class Diagram

4.2.7 Component Diagram

A component diagram in Unified Modeling Language (UML) is a type of structural diagram that illustrates the organization and dependencies between components in a software system. Components are modular parts or building blocks of a system, and a component diagram provides a high-level view of how these components interact and are organized within the system. These diagrams are particularly useful for showing the architecture of a system, highlighting the relationships between components, and helping with system design and development.
Component diagrams are valuable for system architects, software designers, and developers as they provide a visual representation of the system's structure and help in understanding the relationships and interactions between different parts of the system. They can aid in designing scalable, maintainable, and modular software systems.

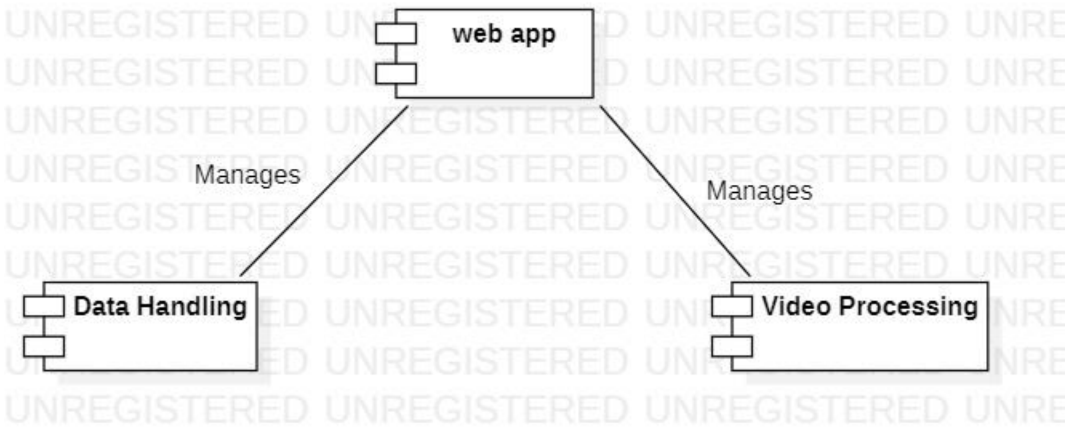


Figure 4.2.7.1 Component Diagram

4.3 Technologies Used

1. ReactJS:
   - The core front-end technology driving the user interface and interactive components of the movie app.
   - ReactJS is known for its component-based architecture, allowing the development of modular and reusable UI elements.

2. TMDB API (The Movie Database API):
   - Utilized for fetching and integrating movie-related data, including details, genres, and trailers, into the application.
   - TMDB API serves as a valuable external data source, providing real-time information about movies.

3. Firebase Google Authentication:
   - Implemented for user authentication, allowing users to sign in securely using their Google accounts.
   - Firebase provides a robust and scalable authentication system, enhancing the app's security and user management.

CHAPTER – 5

5. IMPLEMENTATION

5.1 Detailed Explanation of Project Execution

The execution of the React movie app project is a multi-faceted process that involves a series of well-defined phases, each contributing to the overall development and deployment of the platform. The project execution can be divided into several key steps to ensure a systematic and organized approach.

1. Project Initiation:

The project initiation phase marks the beginning of the execution process. During this stage, the project team is formed, roles and responsibilities are assigned, and a project plan is established. The initial planning includes defining the project's scope, objectives, and deliverables, as well as setting timelines and budgetary considerations. Key stakeholders and team members are introduced to the project's goals and objectives, and the project's feasibility is assessed.

2. Requirements Gathering:

In this phase, the project team collaborates with stakeholders to gather detailed requirements for the React movie app. This includes defining user stories, feature specifications, and functionality expectations. The requirements gathering process involves discussions, surveys, and market research to ensure a comprehensive understanding of user needs and industry trends.

3. Design and Architecture:

The design and architecture phase focuses on creating the blueprint for the application. The project team outlines the platform's structure, user interface design, and data architecture. This includes wireframing, prototyping, and creating design mockups. Additionally, the architectural decisions are made regarding the technology stack, database structure, and integration points with external services like the TMDB API and Firebase.

4. Development:

The development phase is the core of the project execution, where the React movie app is built according to the specifications and design established in the previous phases. Features such as user authentication, movie catalog integration, search functionality, and user personalization are implemented in this phase.

5. Quality Assurance and Testing:

Quality assurance is a continuous process throughout the execution of the project. Dedicated QA teams conduct various tests, including unit testing, integration testing, usability testing, security testing, and performance testing. The goal is to identify and address bugs, security vulnerabilities, and usability issues. This phase ensures that the application is robust, secure, user-friendly.

6. Deployment:

Once the development and testing phases are completed, the React movie app is deployed to a staging environment for final evaluation and user acceptance testing. This staging environment allows selected users to review and provide feedback on the application. Any final adjustments and bug fixes are implemented before moving to the production environment.

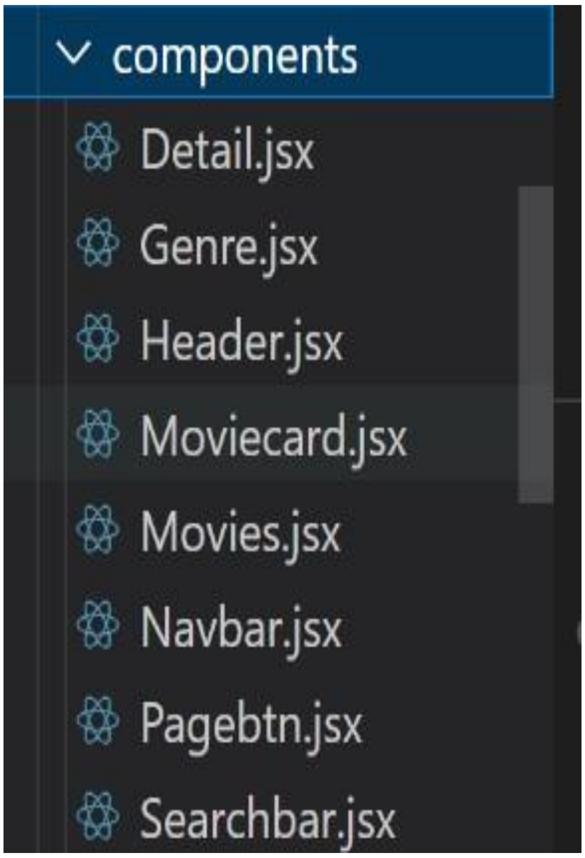
7. Production Release:

With approval from stakeholders and users, the React movie app is released to the environment and made accessible to the public. This phase involves monitoring server performance, ensuring data security, and preparing for user traffic. Additionally, the project team implements continuous monitoring and support processes to address any unforeseen issues that may arise in the live environment.

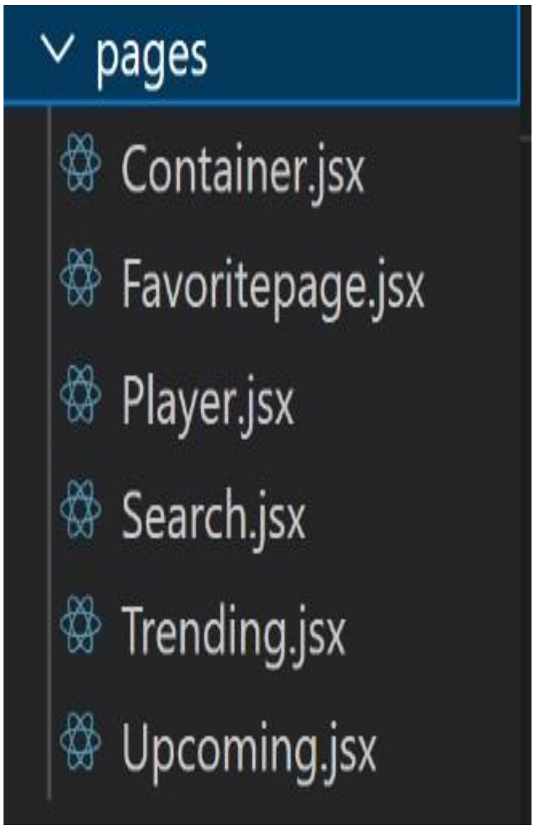
8. User Training and Documentation:

User training materials and documentation are prepared to assist users in navigating the application. This includes creating user guides, FAQs, and instructional videos to ensure that users can maximize the platform's features and capabilities.

5.2 Screenshots

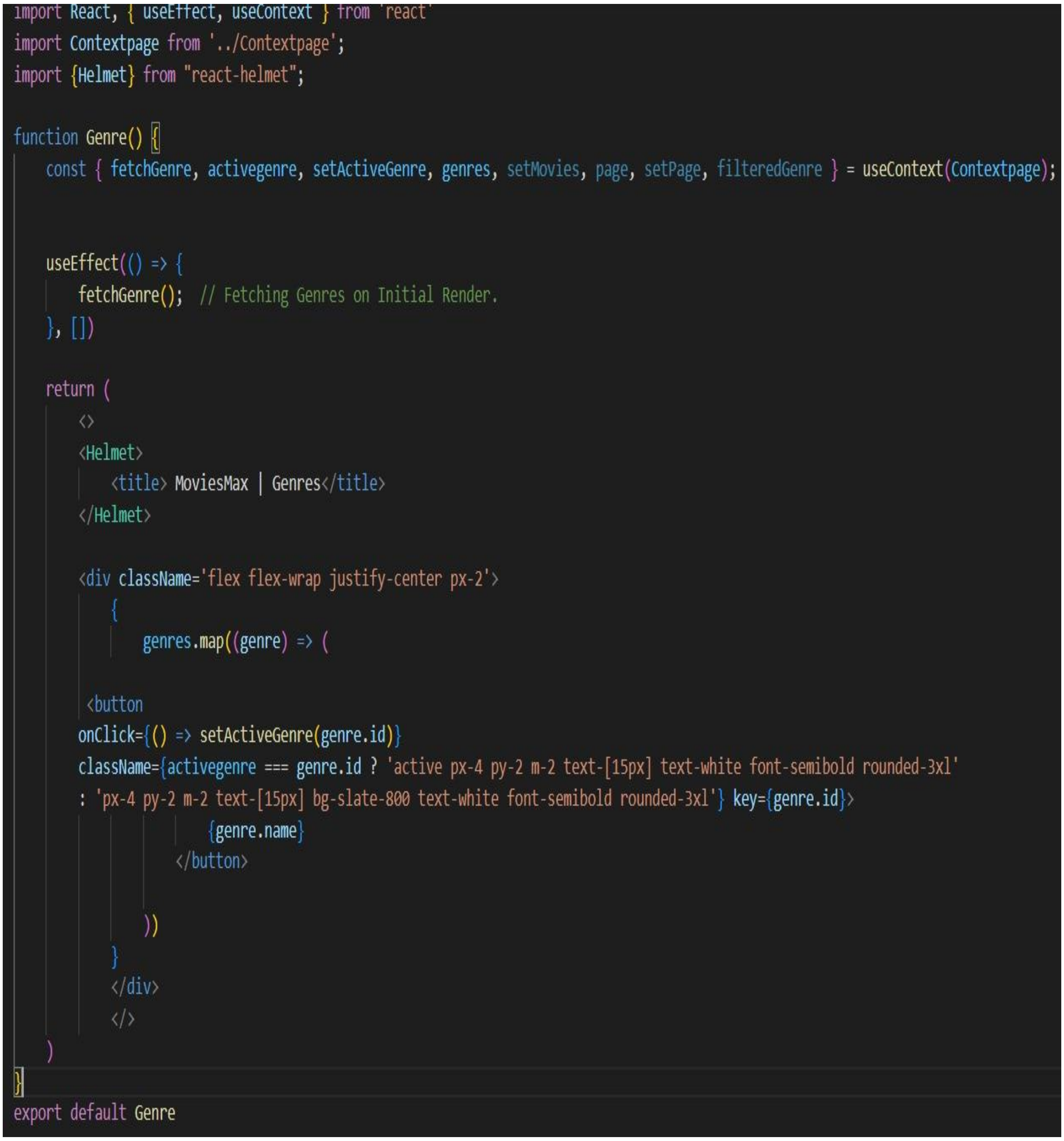


components
Detail.jsx
Genre.jsx
Header.jsx
Moviecard.jsx
Movies.jsx
Navbar.jsx
Pagebtn.jsx
Searchbar.jsx



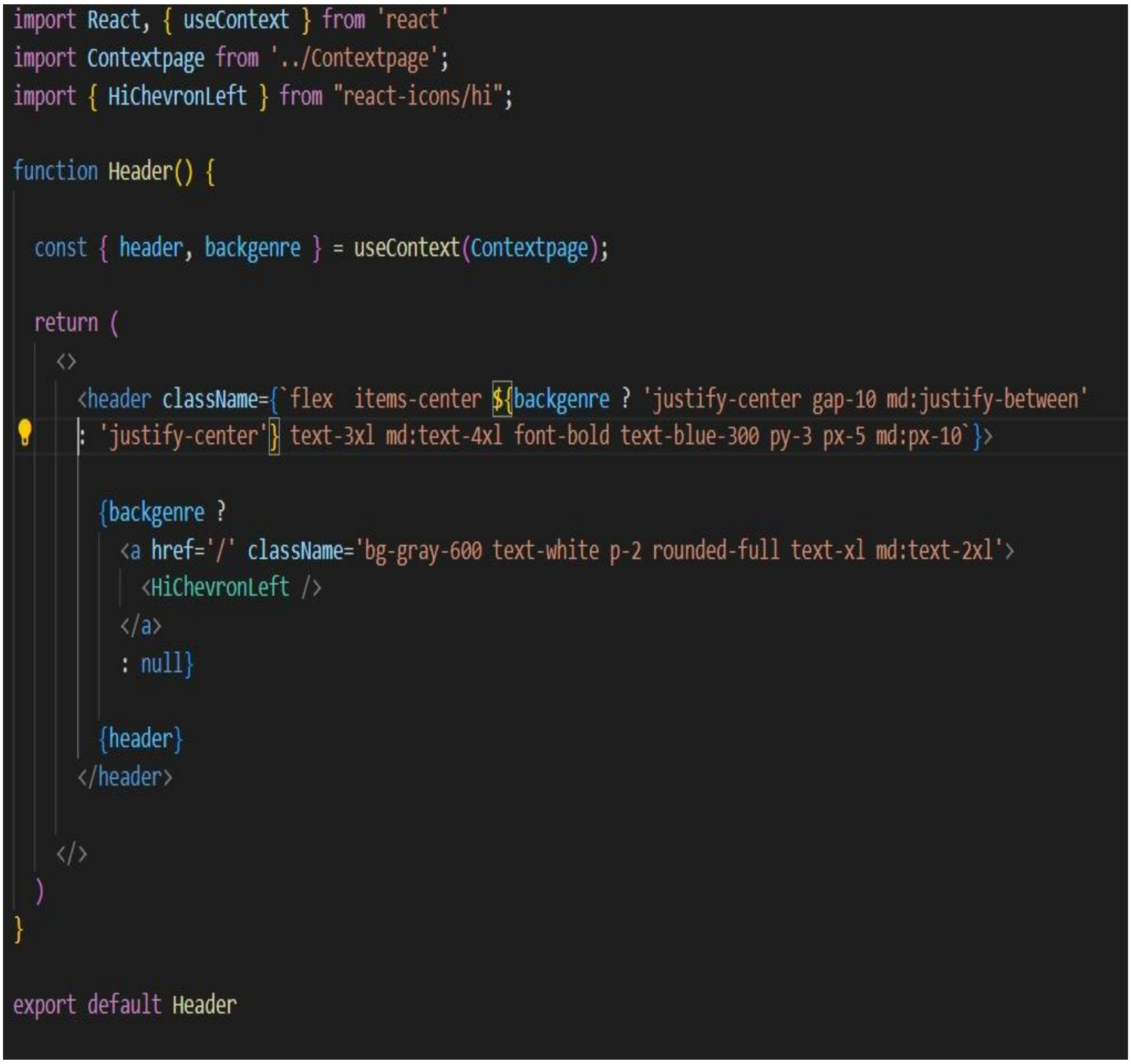
pages
Container.jsx
Favoritepage.jsx
Player.jsx
Search.jsx
Trending.jsx
Upcoming.jsx

Genre.jsx:



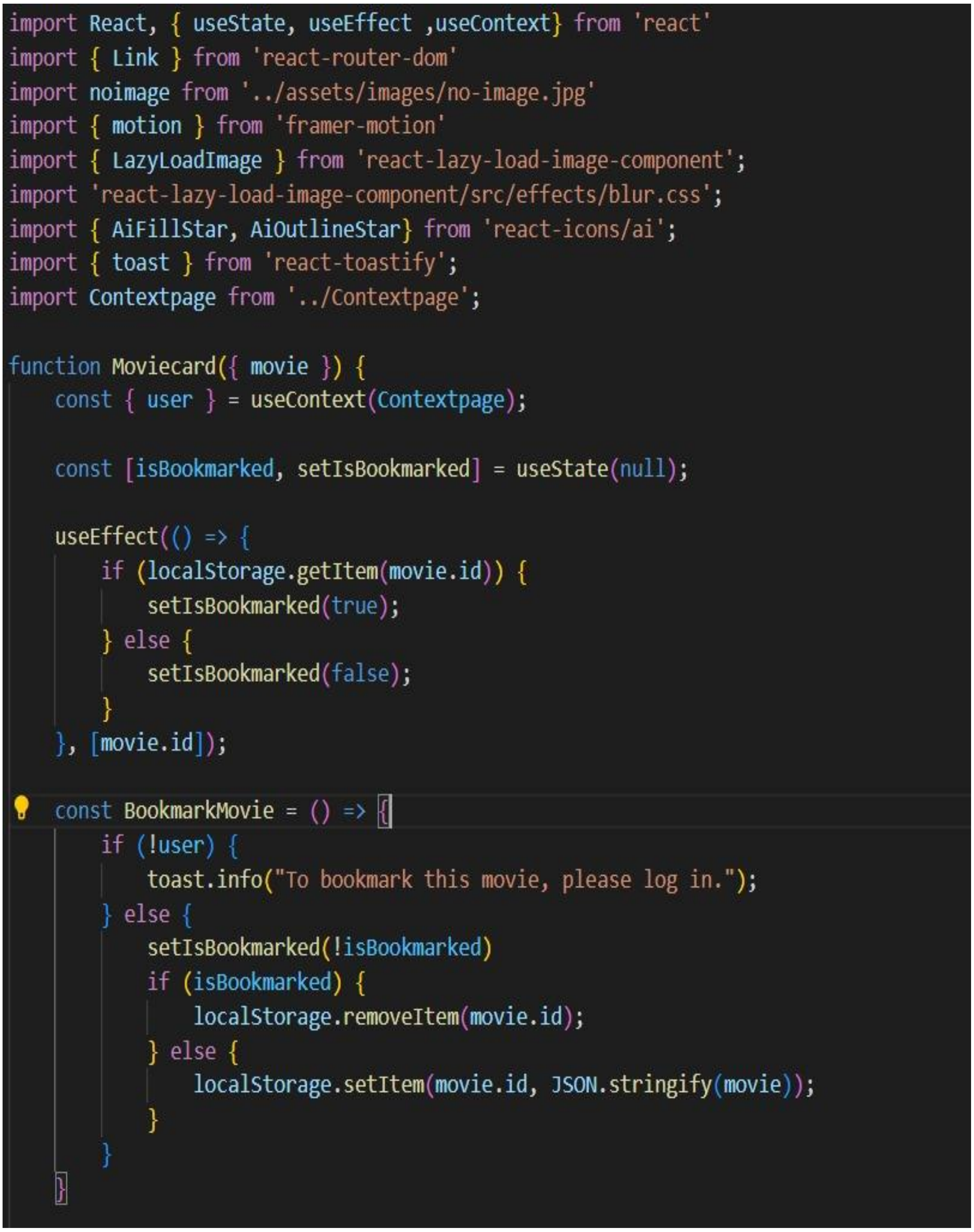
```
import React, { useEffect, useContext } from 'react'
import Contextpage from '../Contextpage';
import {Helmet} from "react-helmet";

function Genre() {
    const { fetchGenre, activegenre, setActiveGenre, genres, setMovies, page, setPage, filteredGenre } = useContext(Contextpage);


    useEffect(() => {
        fetchGenre();  // Fetching Genres on Initial Render.
    }, [])

    return (
        <>
        <Helmet>
            <title> MoviesMax | Genres</title>
        </Helmet>

        <div className='flex flex-wrap justify-center px-2'>
            {
                genres.map((genre) => (

         <button
        onClick={() => setActiveGenre(genre.id)}
        className={activegenre === genre.id ? 'active px-4 py-2 m-2 text-[15px] text-white font-semibold rounded-3xl'
        : 'px-4 py-2 m-2 text-[15px] bg-slate-800 text-white font-semibold rounded-3xl'} key={genre.id}>
                        {genre.name}
                    </button>

                ))
            }
            </div>
            </>
    )
}
export default Genre
```

Header.jsx:

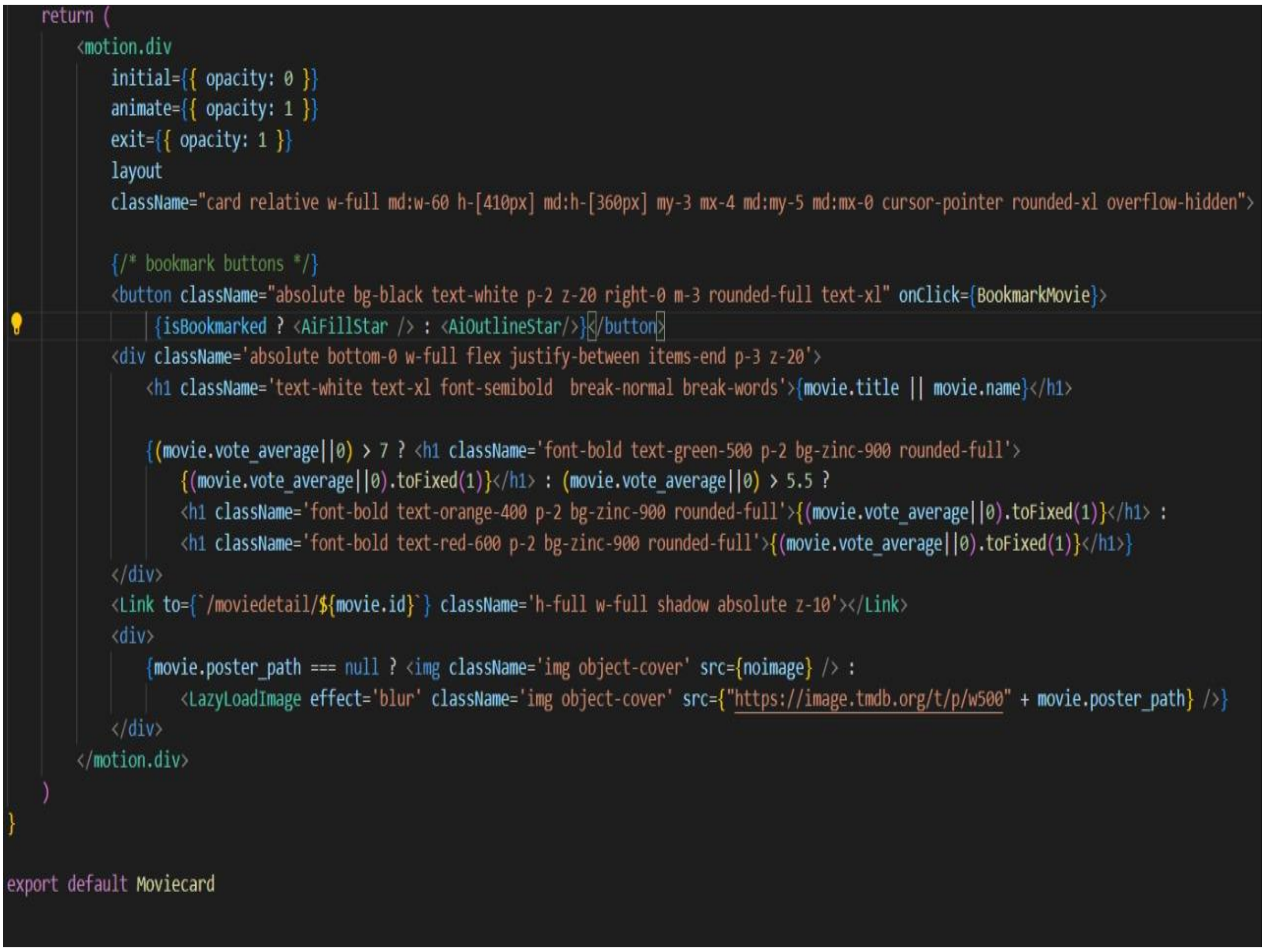


```jsx
import React, { useContext } from 'react'
import Contextpage from '../Contextpage';
import { HiChevronLeft } from "react-icons/hi";

function Header() {

  const { header, backgenre } = useContext(Contextpage);

  return (
    <>
      <header className={`flex  items-center ${backgenre ? 'justify-center gap-10 md:justify-between'
      : 'justify-center'} text-3xl md:text-4xl font-bold text-blue-300 py-3 px-5 md:px-10`}>

        {backgenre ?
          <a href='/' className='bg-gray-600 text-white p-2 rounded-full text-xl md:text-2xl'>
            <HiChevronLeft />
          </a>
          : null}

        {header}
      </header>

    </>
  )
}

export default Header
```

Moviecard.jsx:

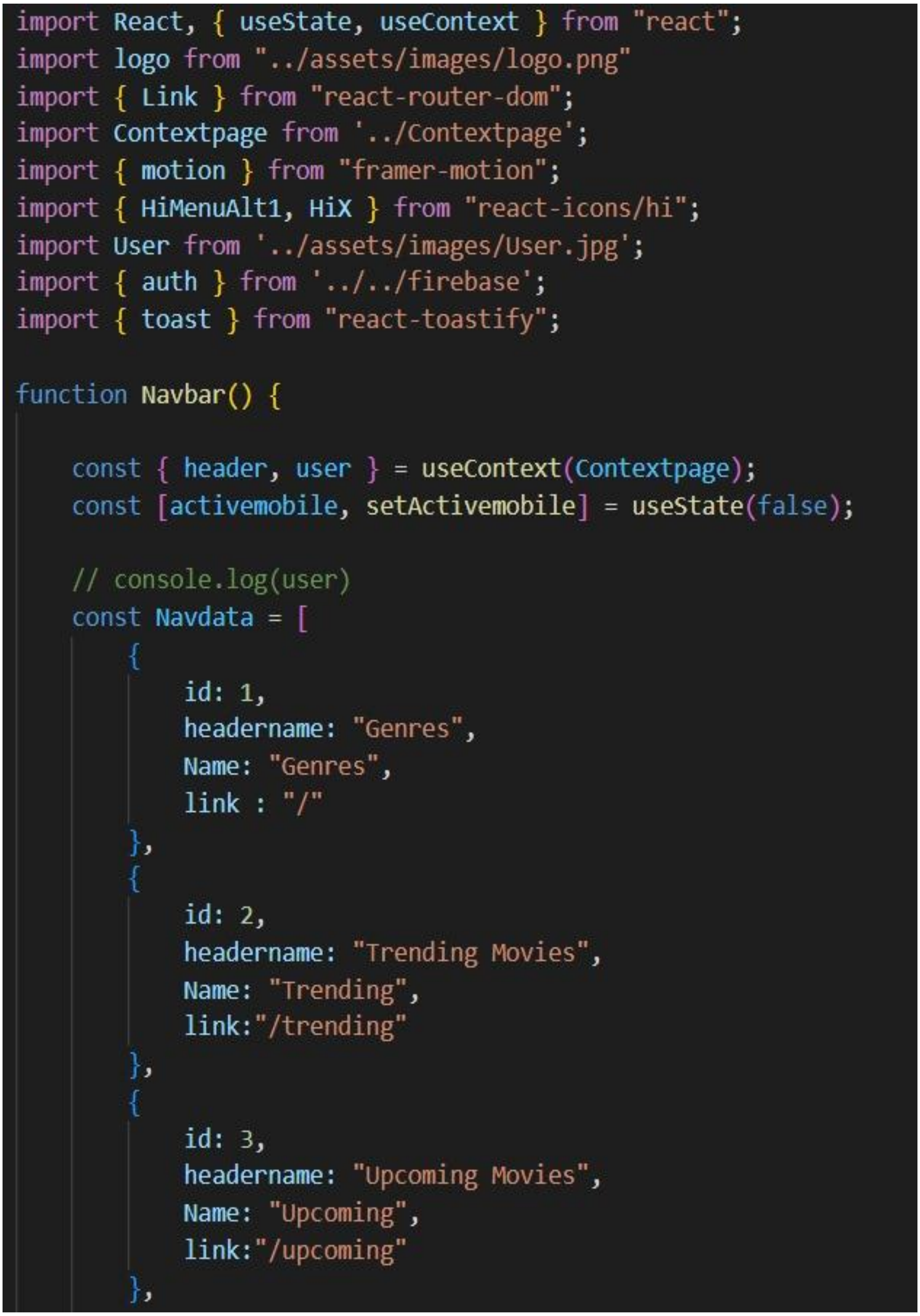


```jsx
import React, { useState, useEffect ,useContext} from 'react'
import { Link } from 'react-router-dom'
import noimage from '../assets/images/no-image.jpg'
import { motion } from 'framer-motion'
import { LazyLoadImage } from 'react-lazy-load-image-component';
import 'react-lazy-load-image-component/src/effects/blur.css';
import { AiFillStar, AiOutlineStar} from 'react-icons/ai';
import { toast } from 'react-toastify';
import Contextpage from '../Contextpage';

function Moviecard({ movie }) {
    const { user } = useContext(Contextpage);

    const [isBookmarked, setIsBookmarked] = useState(null);

    useEffect(() => {
        if (localStorage.getItem(movie.id)) {
            setIsBookmarked(true);
        } else {
            setIsBookmarked(false);
        }
    }, [movie.id]);

    const BookmarkMovie = () => {
        if (!user) {
            toast.info("To bookmark this movie, please log in.");
        } else {
            setIsBookmarked(!isBookmarked)
            if (isBookmarked) {
                localStorage.removeItem(movie.id);
            } else {
                localStorage.setItem(movie.id, JSON.stringify(movie));
            }
        }
    }
```

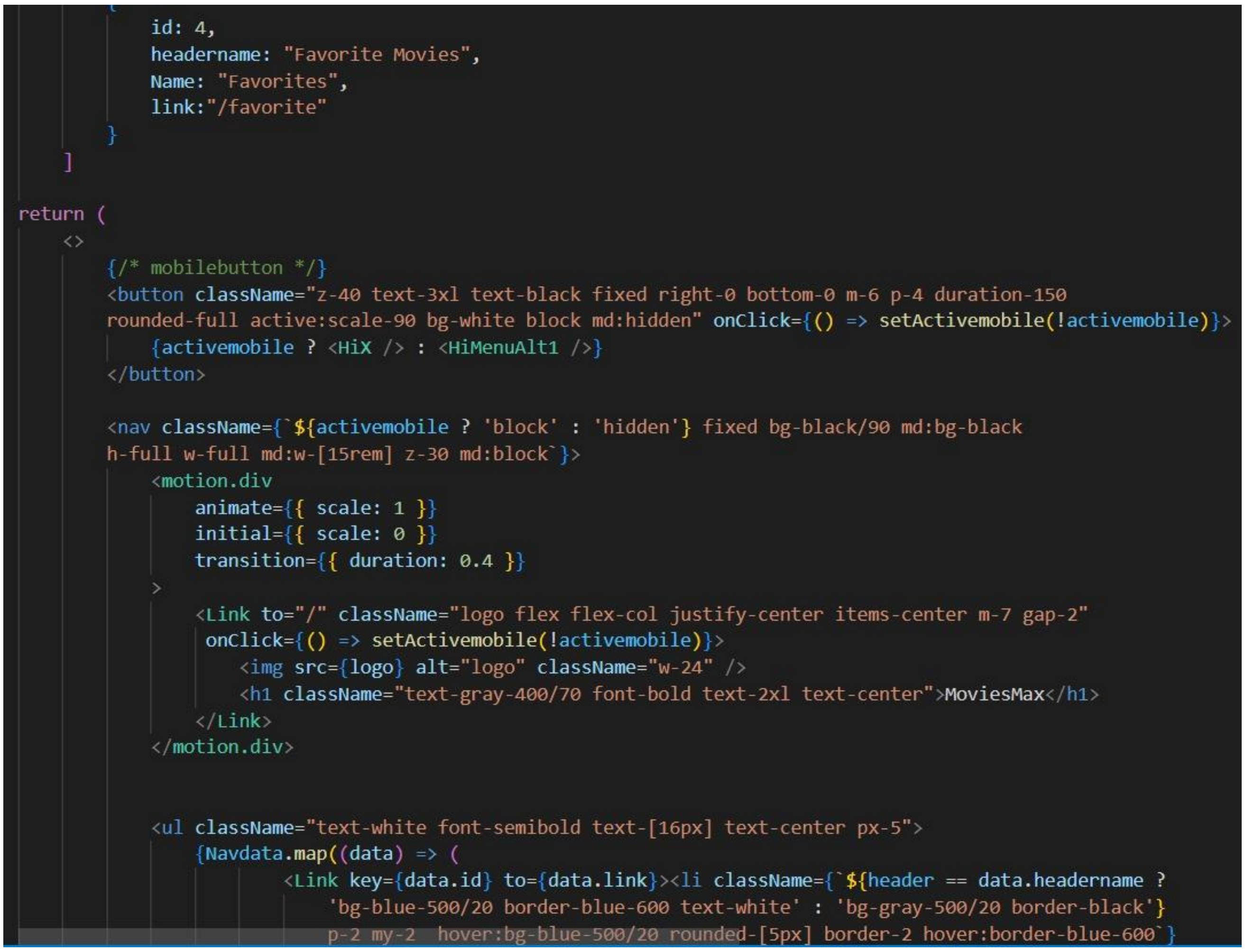


```
    return (
        <motion.div
            initial={{ opacity: 0 }}
            animate={{ opacity: 1 }}
            exit={{ opacity: 1 }}
            layout
            className="card relative w-full md:w-60 h-[410px] md:h-[360px] my-3 mx-4 md:my-5 md:mx-0 cursor-pointer rounded-xl overflow-hidden">

            {/* bookmark buttons */}
            <button className="absolute bg-black text-white p-2 z-20 right-0 m-3 rounded-full text-xl" onClick={BookmarkMovie}>
                {isBookmarked ? <AiFillStar /> : <AiOutlineStar/>}</button>
            <div className='absolute bottom-0 w-full flex justify-between items-end p-3 z-20'>
                <h1 className='text-white text-xl font-semibold  break-normal break-words'>{movie.title || movie.name}</h1>

                {(movie.vote_average||0) > 7 ? <h1 className='font-bold text-green-500 p-2 bg-zinc-900 rounded-full'>
                    {(movie.vote_average||0).toFixed(1)}</h1> : (movie.vote_average||0) > 5.5 ?
                    <h1 className='font-bold text-orange-400 p-2 bg-zinc-900 rounded-full'>{(movie.vote_average||0).toFixed(1)}</h1> :
                    <h1 className='font-bold text-red-600 p-2 bg-zinc-900 rounded-full'>{(movie.vote_average||0).toFixed(1)}</h1>}
            </div>
            <Link to={`/moviedetail/${movie.id}`} className='h-full w-full shadow absolute z-10'></Link>
            <div>
                {movie.poster_path === null ? <img className='img object-cover' src={noimage} /> :
                    <LazyLoadImage effect='blur' className='img object-cover' src={"https://image.tmdb.org/t/p/w500" + movie.poster_path} />}
            </div>
        </motion.div>
    )
}

export default Moviecard
```

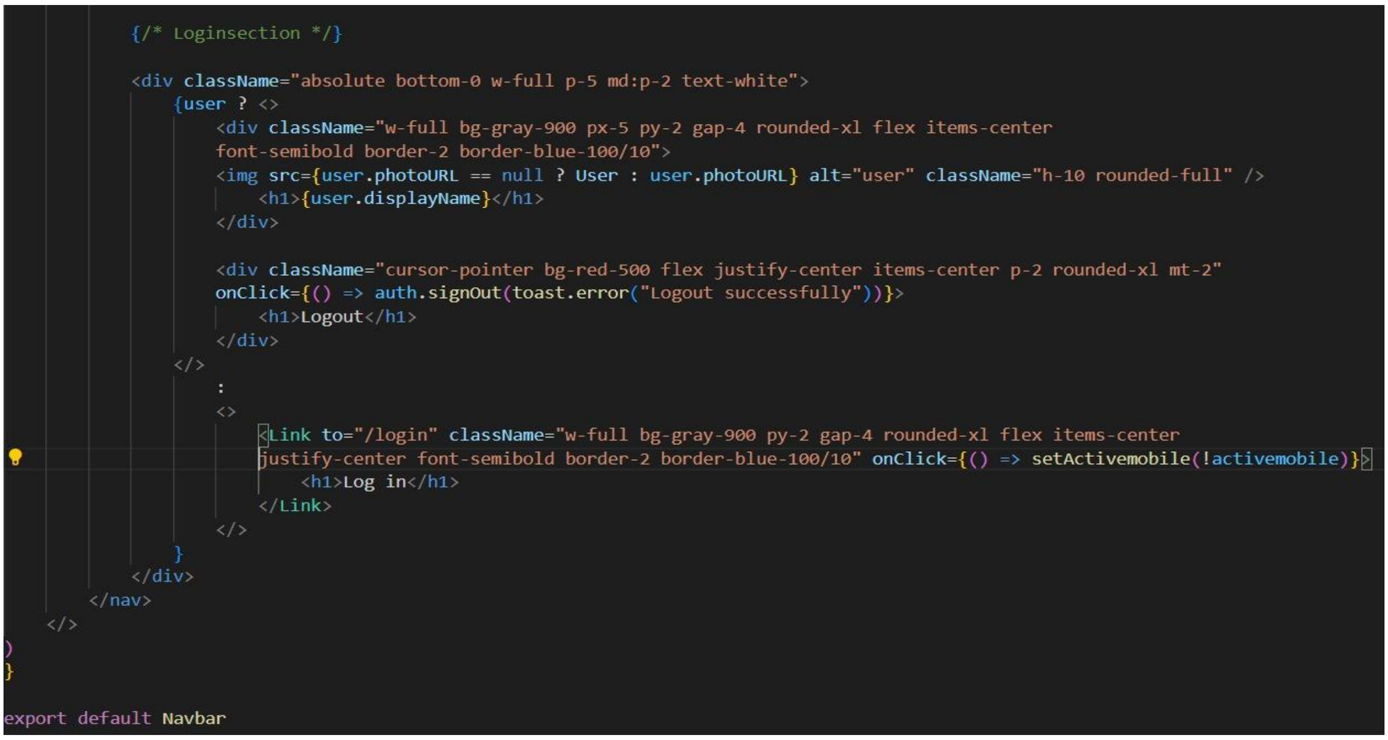
NavBar.jsx:



```
import React, { useState, useContext } from "react";
import logo from "../assets/images/logo.png"
import { Link } from "react-router-dom";
import Contextpage from '../Contextpage';
import { motion } from "framer-motion";
import { HiMenuAlt1, HiX } from "react-icons/hi";
import User from '../assets/images/User.jpg';
import { auth } from '../../firebase';
import { toast } from "react-toastify";

function Navbar() {

    const { header, user } = useContext(Contextpage);
    const [activemobile, setActivemobile] = useState(false);

    // console.log(user)
    const Navdata = [
        {
            id: 1,
            headername: "Genres",
            Name: "Genres",
            link : "/"
        },
        {
            id: 2,
            headername: "Trending Movies",
            Name: "Trending",
            link:"/trending"
        },
        {
            id: 3,
            headername: "Upcoming Movies",
            Name: "Upcoming",
            link:"/upcoming"
        },
```

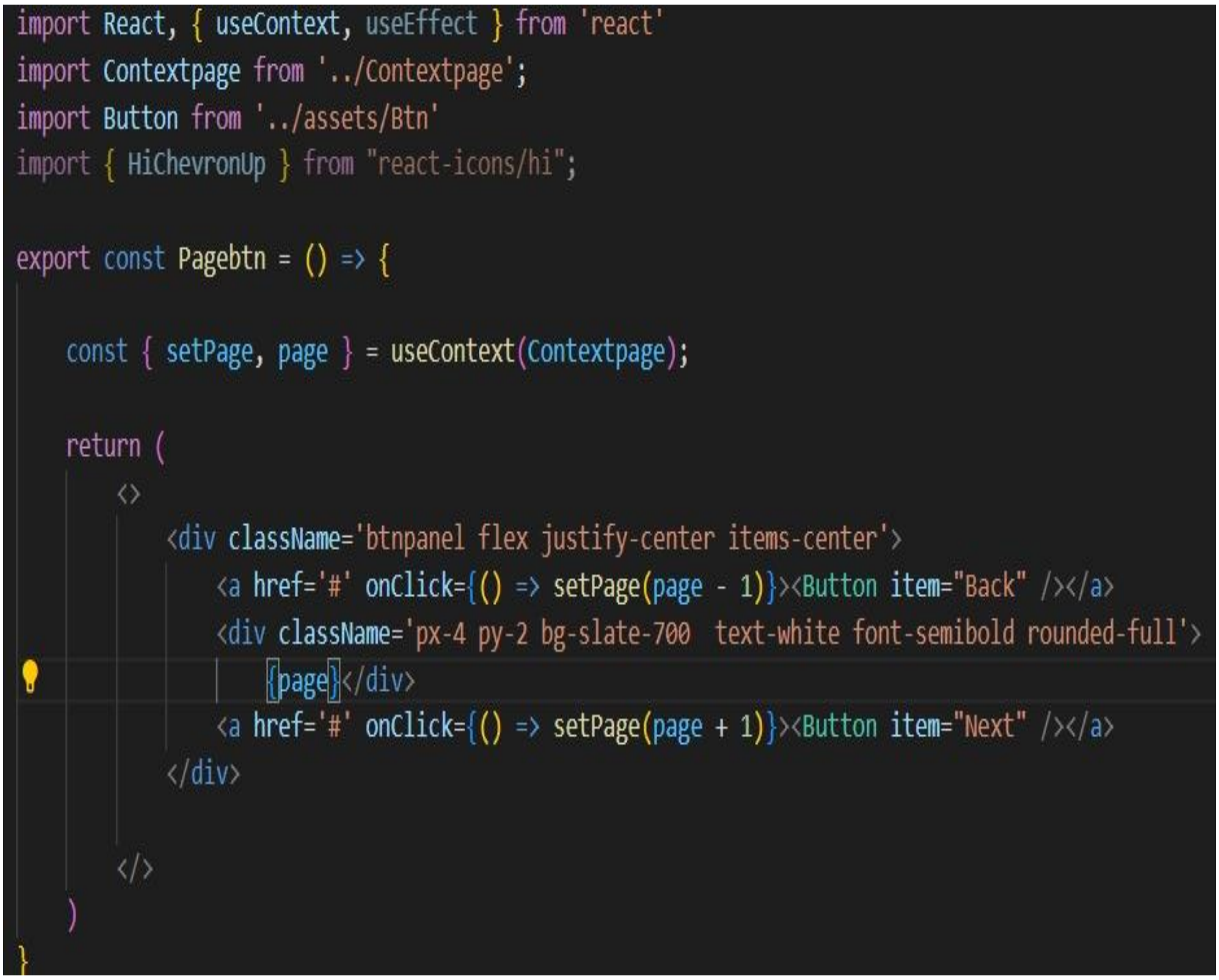


```
            id: 4,
            headername: "Favorite Movies",
            Name: "Favorites",
            link:"/favorite"
        }
    ]

return (
    <>
        {/* mobilebutton */}
        <button className="z-40 text-3xl text-black fixed right-0 bottom-0 m-6 p-4 duration-150
        rounded-full active:scale-90 bg-white block md:hidden" onClick={() => setActivemobile(!activemobile)}>
            {activemobile ? <HiX /> : <HiMenuAlt1 />}
        </button>

        <nav className={`${activemobile ? 'block' : 'hidden'} fixed bg-black/90 md:bg-black
        h-full w-full md:w-[15rem] z-30 md:block`}>
            <motion.div
                animate={{ scale: 1 }}
                initial={{ scale: 0 }}
                transition={{ duration: 0.4 }}
            >
                <Link to="/" className="logo flex flex-col justify-center items-center m-7 gap-2"
                 onClick={() => setActivemobile(!activemobile)}>
                    <img src={logo} alt="logo" className="w-24" />
                    <h1 className="text-gray-400/70 font-bold text-2xl text-center">MoviesMax</h1>
                </Link>
            </motion.div>


            <ul className="text-white font-semibold text-[16px] text-center px-5">
                {Navdata.map((data) => (
                        <Link key={data.id} to={data.link}><li className={`${header == data.headername ?
                            'bg-blue-500/20 border-blue-600 text-white' : 'bg-gray-500/20 border-black'}
                            p-2 my-2  hover:bg-blue-500/20 rounded-[5px] border-2 hover:border-blue-600`}
```



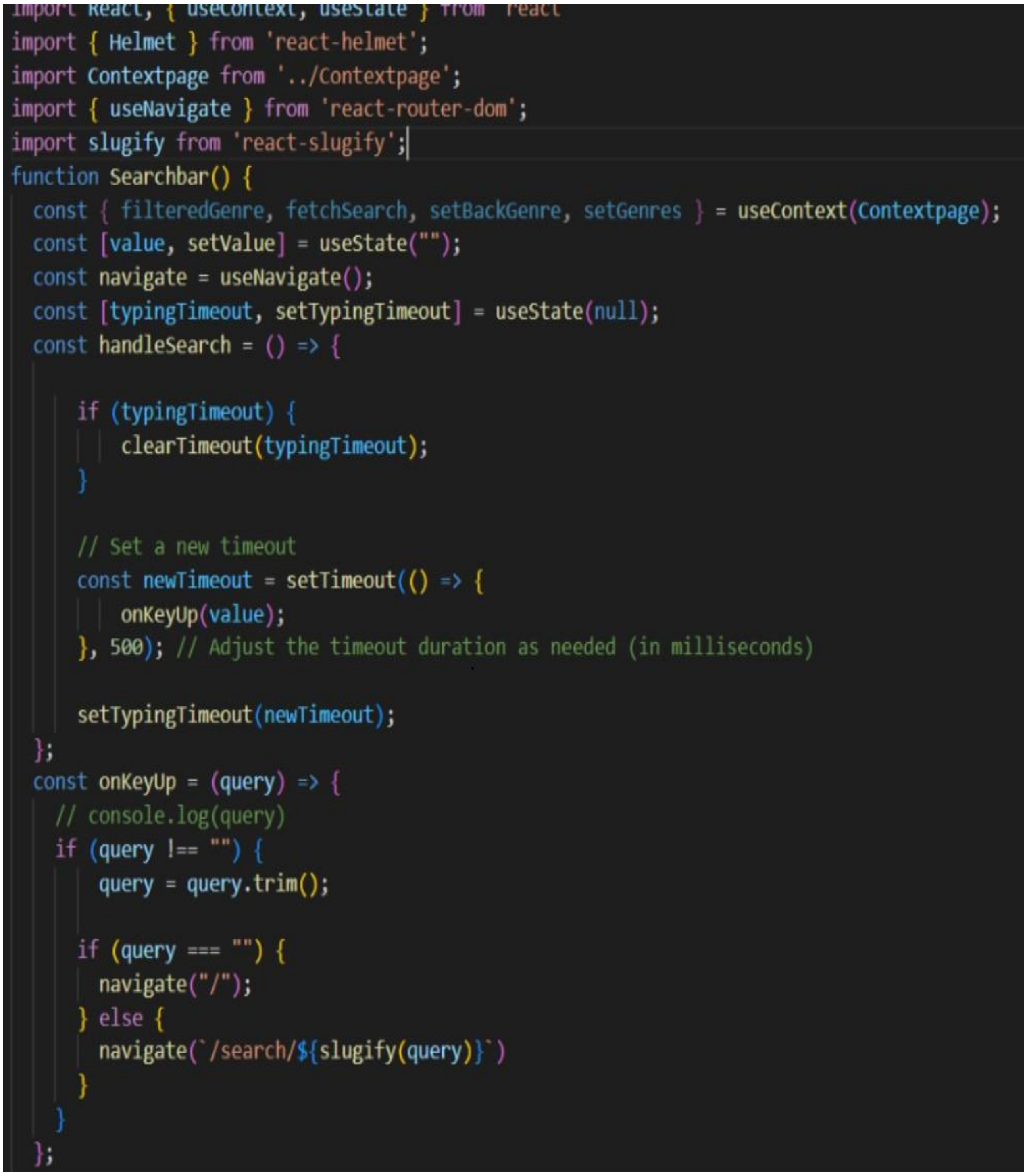
```
            {/* Loginsection */}

            <div className="absolute bottom-0 w-full p-5 md:p-2 text-white">
                {user ? <>
                    <div className="w-full bg-gray-900 px-5 py-2 gap-4 rounded-xl flex items-center
                    font-semibold border-2 border-blue-100/10">
                    <img src={user.photoURL == null ? User : user.photoURL} alt="user" className="h-10 rounded-full" />
                        <h1>{user.displayName}</h1>
                    </div>

                    <div className="cursor-pointer bg-red-500 flex justify-center items-center p-2 rounded-xl mt-2"
                    onClick={() => auth.signOut(toast.error("Logout successfully"))}>
                        <h1>Logout</h1>
                    </div>
                </>
                    :
                    <>
                        <Link to="/login" className="w-full bg-gray-900 py-2 gap-4 rounded-xl flex items-center
                        justify-center font-semibold border-2 border-blue-100/10" onClick={() => setActivemobile(!activemobile)}>
                            <h1>Log in</h1>
                        </Link>
                    </>
                }
            </div>
        </nav>
    </>
)
}

export default Navbar
```

PageBtn.jsx:

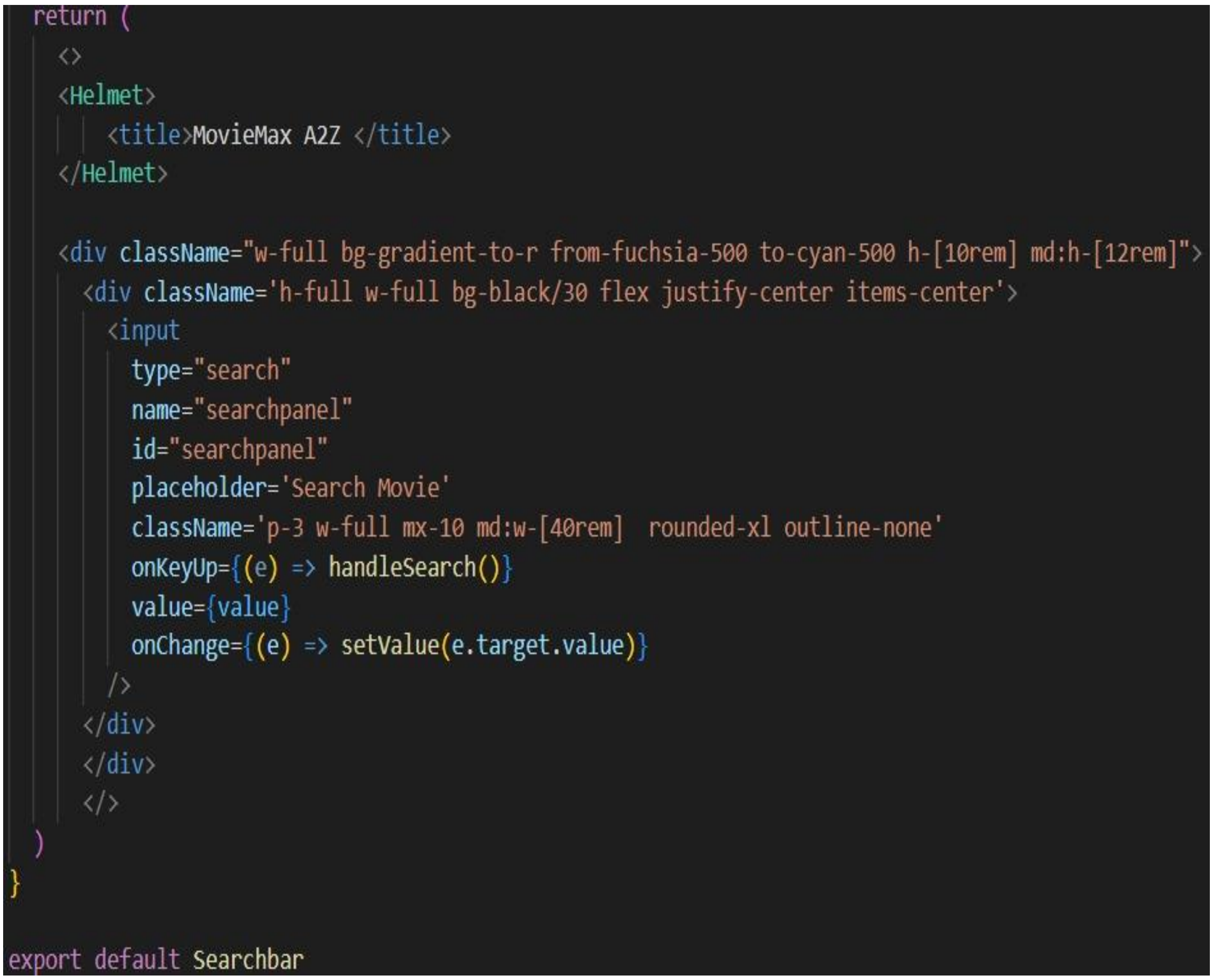


```
import React, { useContext, useEffect } from 'react'
import Contextpage from '../Contextpage';
import Button from '../assets/Btn'
import { HiChevronUp } from "react-icons/hi";

export const Pagebtn = () => {

    const { setPage, page } = useContext(Contextpage);

    return (
        <>
            <div className='btnpanel flex justify-center items-center'>
                <a href='#' onClick={() => setPage(page - 1)}><Button item="Back" /></a>
                <div className='px-4 py-2 bg-slate-700  text-white font-semibold rounded-full'>
                    {page}</div>
                <a href='#' onClick={() => setPage(page + 1)}><Button item="Next" /></a>
            </div>

        </>
    )
}
```

SearchBar.jsx:

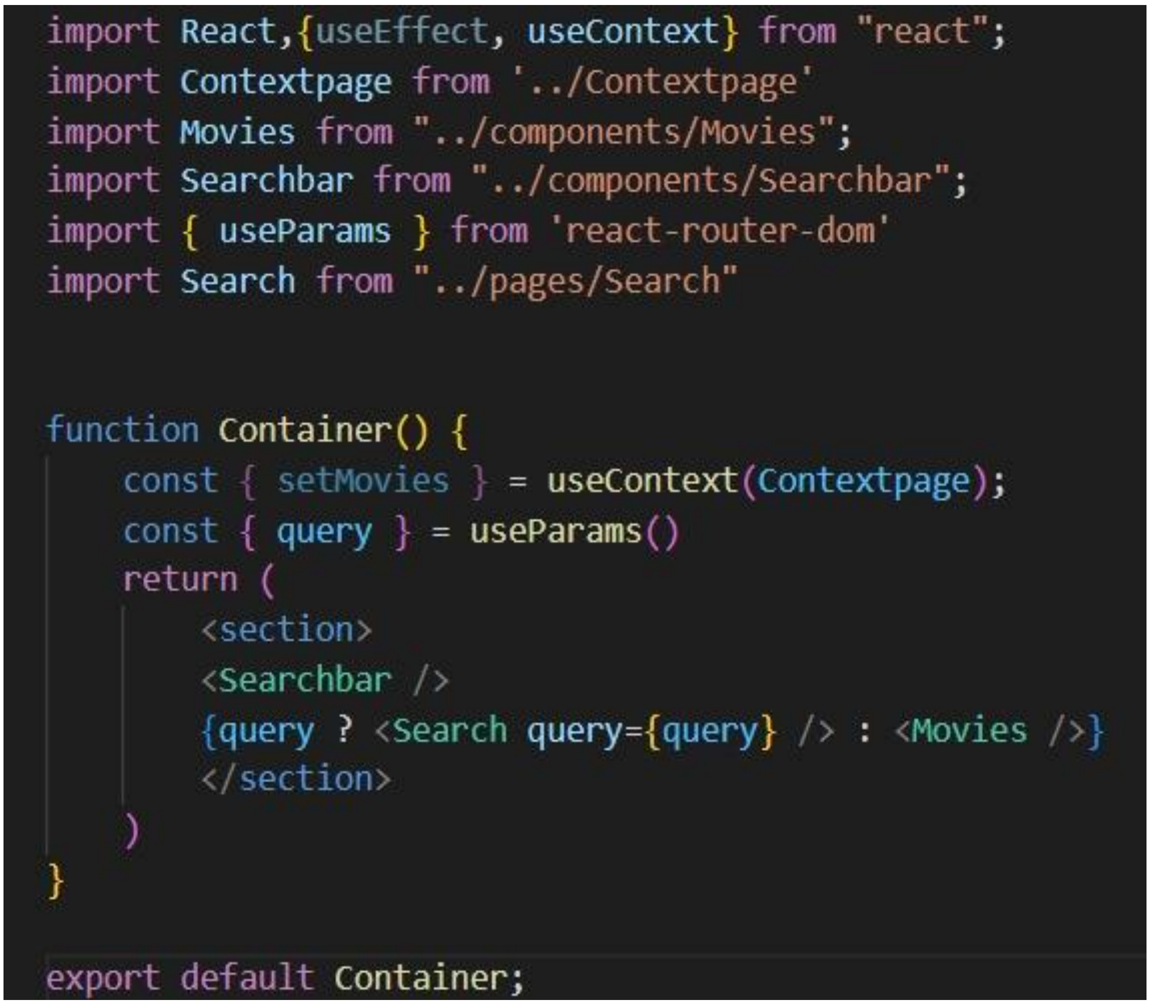


```
import React, { useContext, useState } from 'react'
import { Helmet } from 'react-helmet';
import Contextpage from '../Contextpage';
import { useNavigate } from 'react-router-dom';
import slugify from 'react-slugify';
function Searchbar() {
  const { filteredGenre, fetchSearch, setBackGenre, setGenres } = useContext(Contextpage);
  const [value, setValue] = useState("");
  const navigate = useNavigate();
  const [typingTimeout, setTypingTimeout] = useState(null);
  const handleSearch = () => {

      if (typingTimeout) {
          clearTimeout(typingTimeout);
      }

      // Set a new timeout
      const newTimeout = setTimeout(() => {
          onKeyUp(value);
      }, 500); // Adjust the timeout duration as needed (in milliseconds)

      setTypingTimeout(newTimeout);
  };
  const onKeyUp = (query) => {
    // console.log(query)
    if (query !== "") {
        query = query.trim();

      if (query === "") {
        navigate("/");
      } else {
        navigate(`/search/${slugify(query)}`)
      }
    }
  };
```



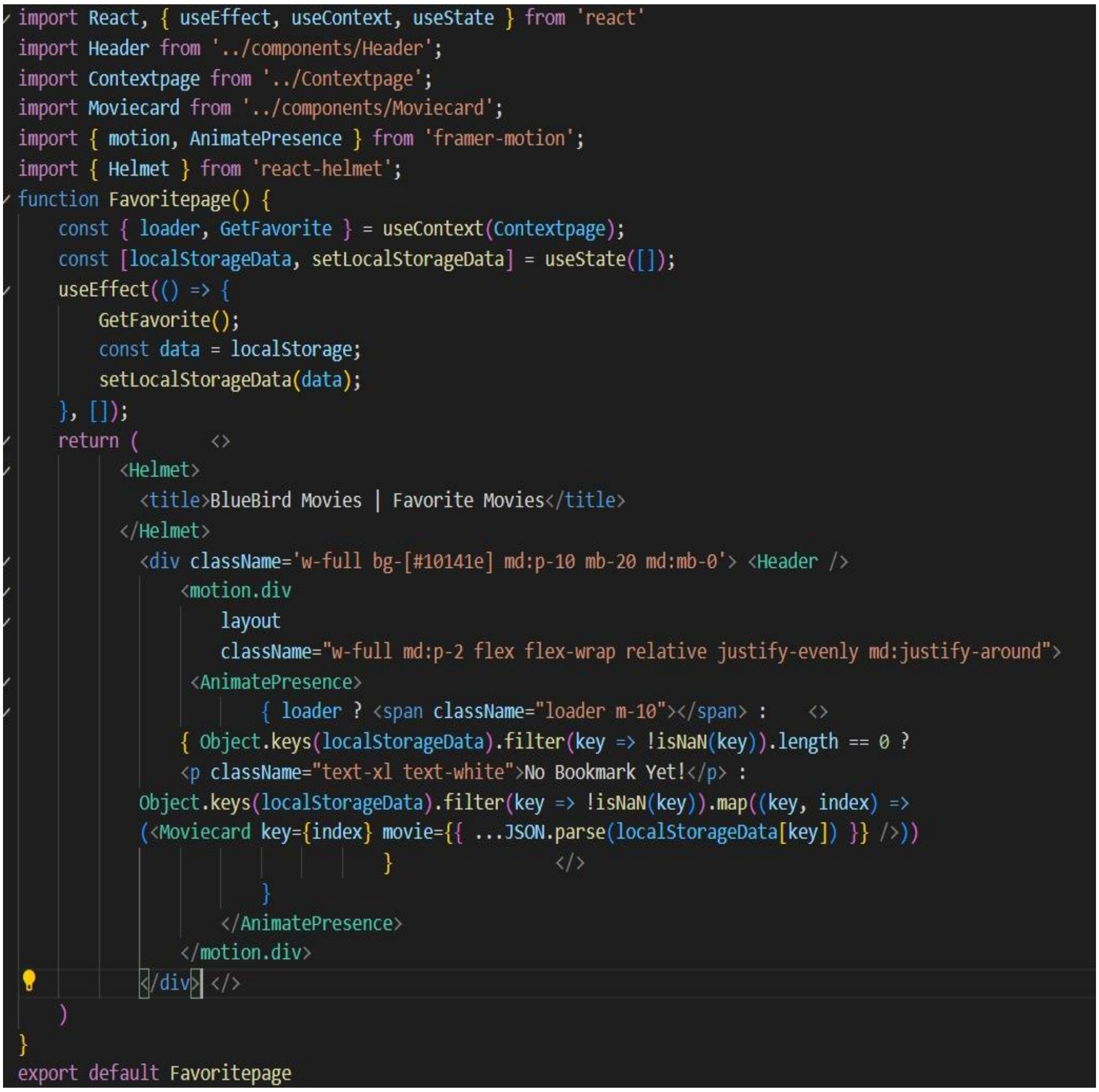
```
  return (
    <>
    <Helmet>
        <title>MovieMax A2Z </title>
    </Helmet>

    <div className="w-full bg-gradient-to-r from-fuchsia-500 to-cyan-500 h-[10rem] md:h-[12rem]">
      <div className='h-full w-full bg-black/30 flex justify-center items-center'>
        <input
          type="search"
          name="searchpanel"
          id="searchpanel"
          placeholder='Search Movie'
          className='p-3 w-full mx-10 md:w-[40rem]  rounded-xl outline-none'
          onKeyUp={(e) => handleSearch()}
          value={value}
          onChange={(e) => setValue(e.target.value)}
        />
      </div>
      </div>
      </>
  )
}

export default Searchbar
```

Container.jsx:



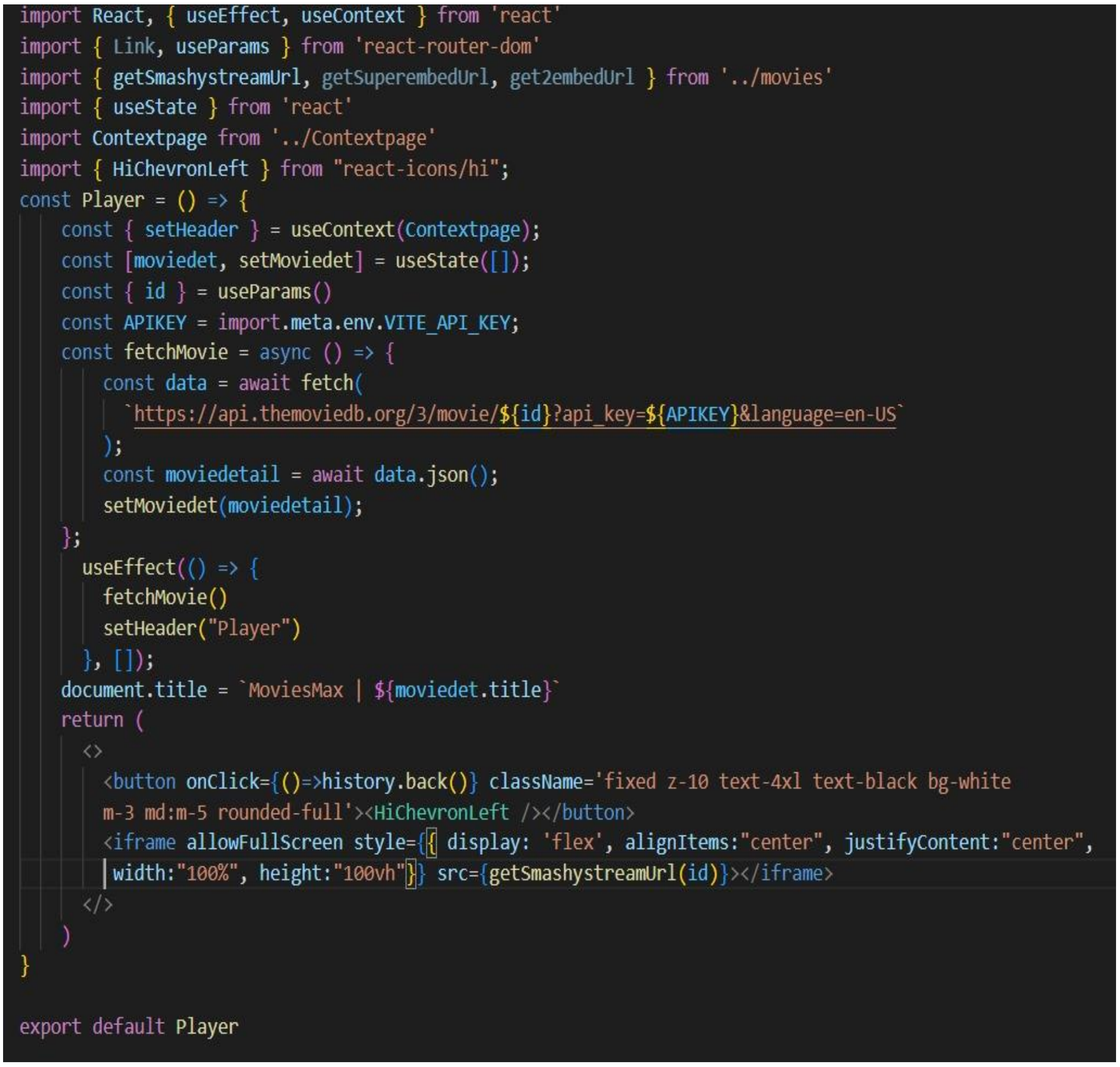
```
import React,{useEffect, useContext} from "react";
import Contextpage from '../Contextpage'
import Movies from "../components/Movies";
import Searchbar from "../components/Searchbar";
import { useParams } from 'react-router-dom'
import Search from "../pages/Search"


function Container() {
    const { setMovies } = useContext(Contextpage);
    const { query } = useParams()
    return (
        <section>
        <Searchbar />
        {query ? <Search query={query} /> : <Movies />}
        </section>
    )
}

export default Container;
```

Favoritepage.jsx:



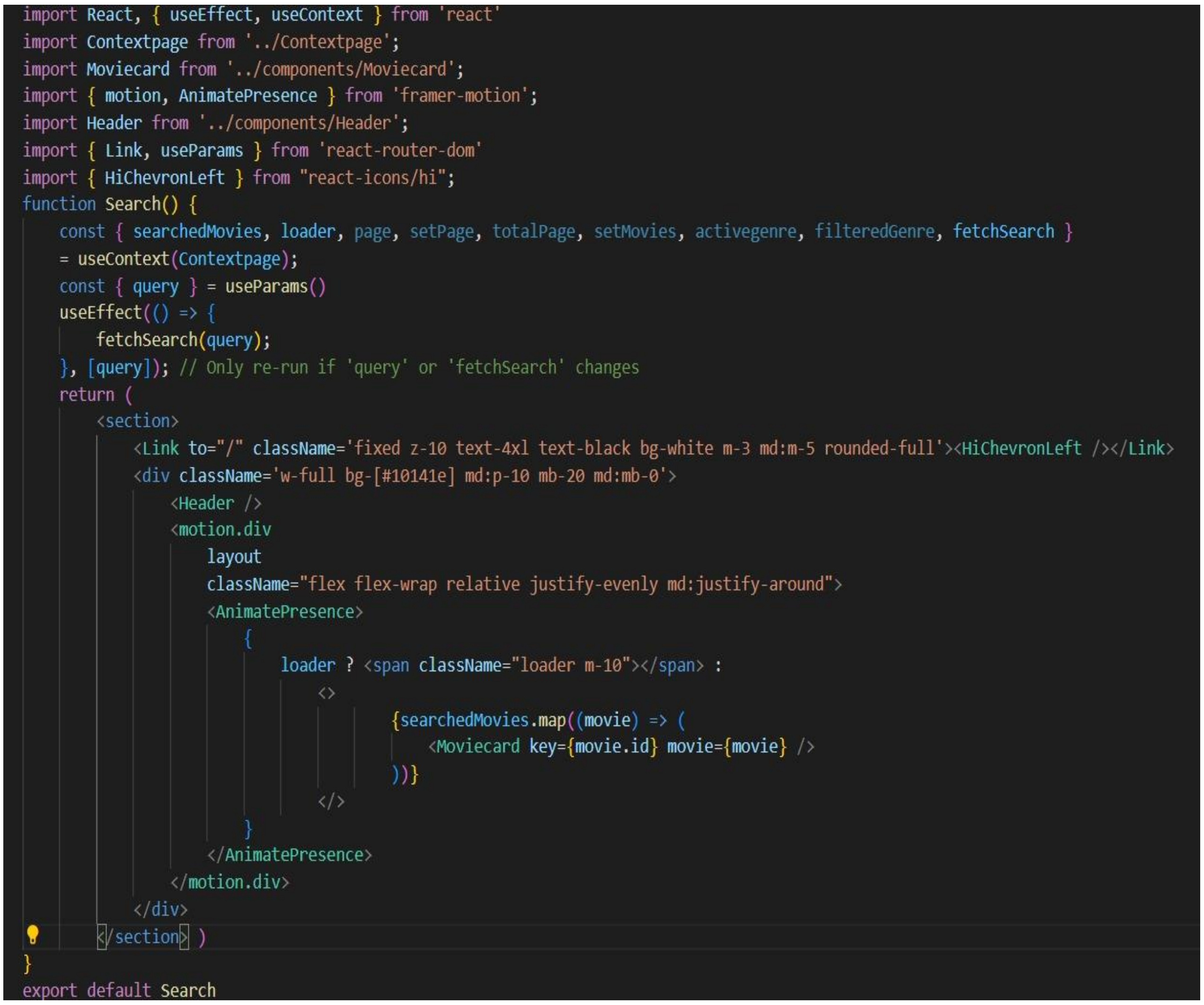
```
import React, { useEffect, useContext, useState } from 'react'
import Header from '../components/Header';
import Contextpage from '../Contextpage';
import Moviecard from '../components/Moviecard';
import { motion, AnimatePresence } from 'framer-motion';
import { Helmet } from 'react-helmet';
function Favoritepage() {
    const { loader, GetFavorite } = useContext(Contextpage);
    const [localStorageData, setLocalStorageData] = useState([]);
    useEffect(() => {
        GetFavorite();
        const data = localStorage;
        setLocalStorageData(data);
    }, []);
    return (       <>
          <Helmet>
            <title>BlueBird Movies | Favorite Movies</title>
          </Helmet>
            <div className='w-full bg-[#10141e] md:p-10 mb-20 md:mb-0'> <Header />
                <motion.div
                    layout
                    className="w-full md:p-2 flex flex-wrap relative justify-evenly md:justify-around">
                 <AnimatePresence>
                        { loader ? <span className="loader m-10"></span> :    <>
                { Object.keys(localStorageData).filter(key => !isNaN(key)).length == 0 ?
                <p className="text-xl text-white">No Bookmark Yet!</p> :
            Object.keys(localStorageData).filter(key => !isNaN(key)).map((key, index) =>
            (<Moviecard key={index} movie={{ ...JSON.parse(localStorageData[key]) }} />))
                            }                </>
                        }
                    </AnimatePresence>
                </motion.div>
            </div> </>
    )
}
export default Favoritepage
```

Player.jsx:



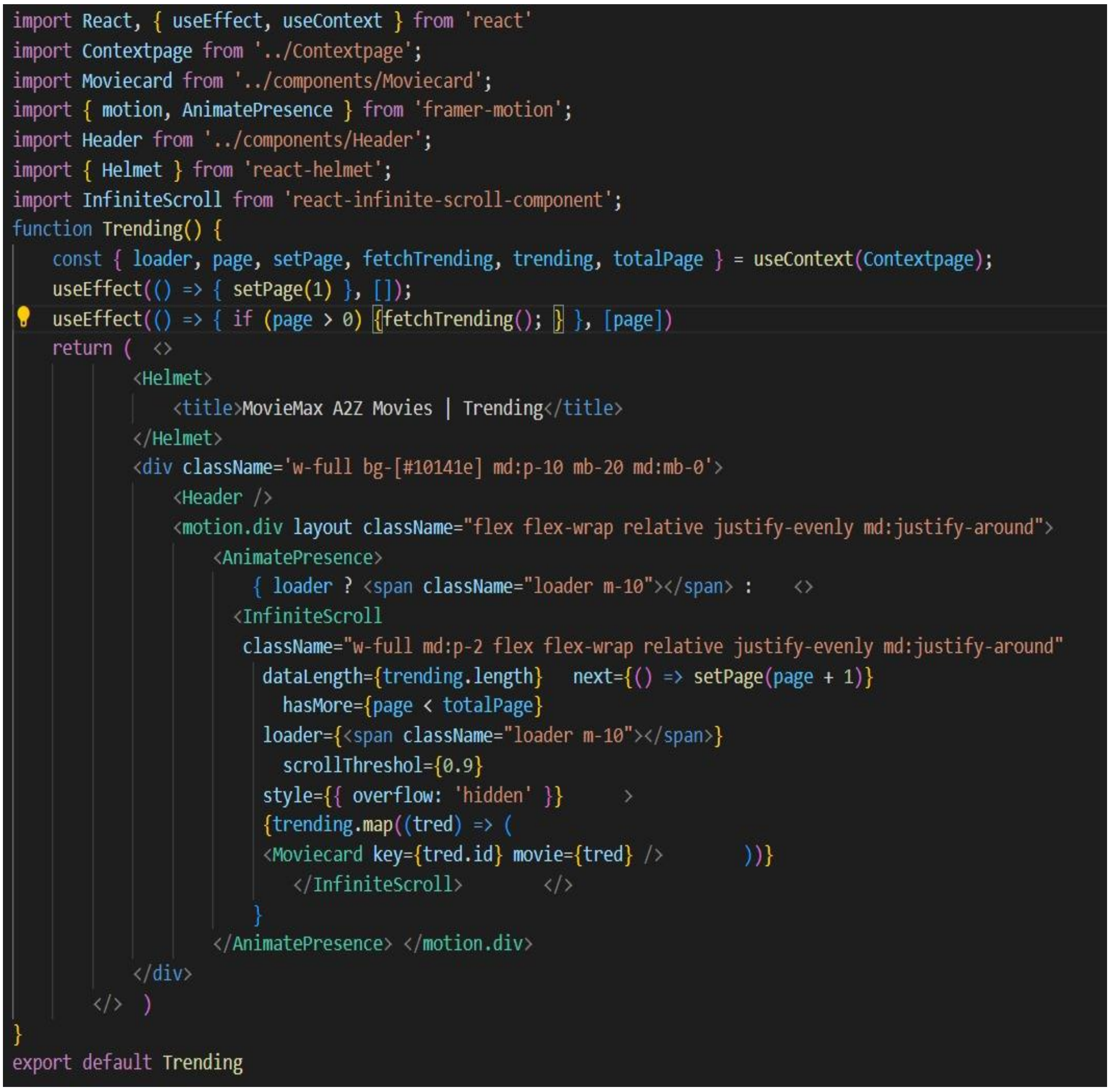
```
import React, { useEffect, useContext } from 'react'
import { Link, useParams } from 'react-router-dom'
import { getSmashystreamUrl, getSuperembedUrl, get2embedUrl } from '../movies'
import { useState } from 'react'
import Contextpage from '../Contextpage'
import { HiChevronLeft } from "react-icons/hi";
const Player = () => {
    const { setHeader } = useContext(Contextpage);
    const [moviedet, setMoviedet] = useState([]);
    const { id } = useParams()
    const APIKEY = import.meta.env.VITE_API_KEY;
    const fetchMovie = async () => {
        const data = await fetch(
          `https://api.themoviedb.org/3/movie/${id}?api_key=${APIKEY}&language=en-US`
        );
        const moviedetail = await data.json();
        setMoviedet(moviedetail);
    };
      useEffect(() => {
        fetchMovie()
        setHeader("Player")
      }, []);
    document.title = `MoviesMax | ${moviedet.title}`
    return (
      <>
        <button onClick={()=>history.back()} className='fixed z-10 text-4xl text-black bg-white
        m-3 md:m-5 rounded-full'><HiChevronLeft /></button>
        <iframe allowFullScreen style={{ display: 'flex', alignItems:"center", justifyContent:"center",
        width:"100%", height:"100vh"}} src={getSmashystreamUrl(id)}></iframe>
      </>
    )
}

export default Player
```

Search.jsx:



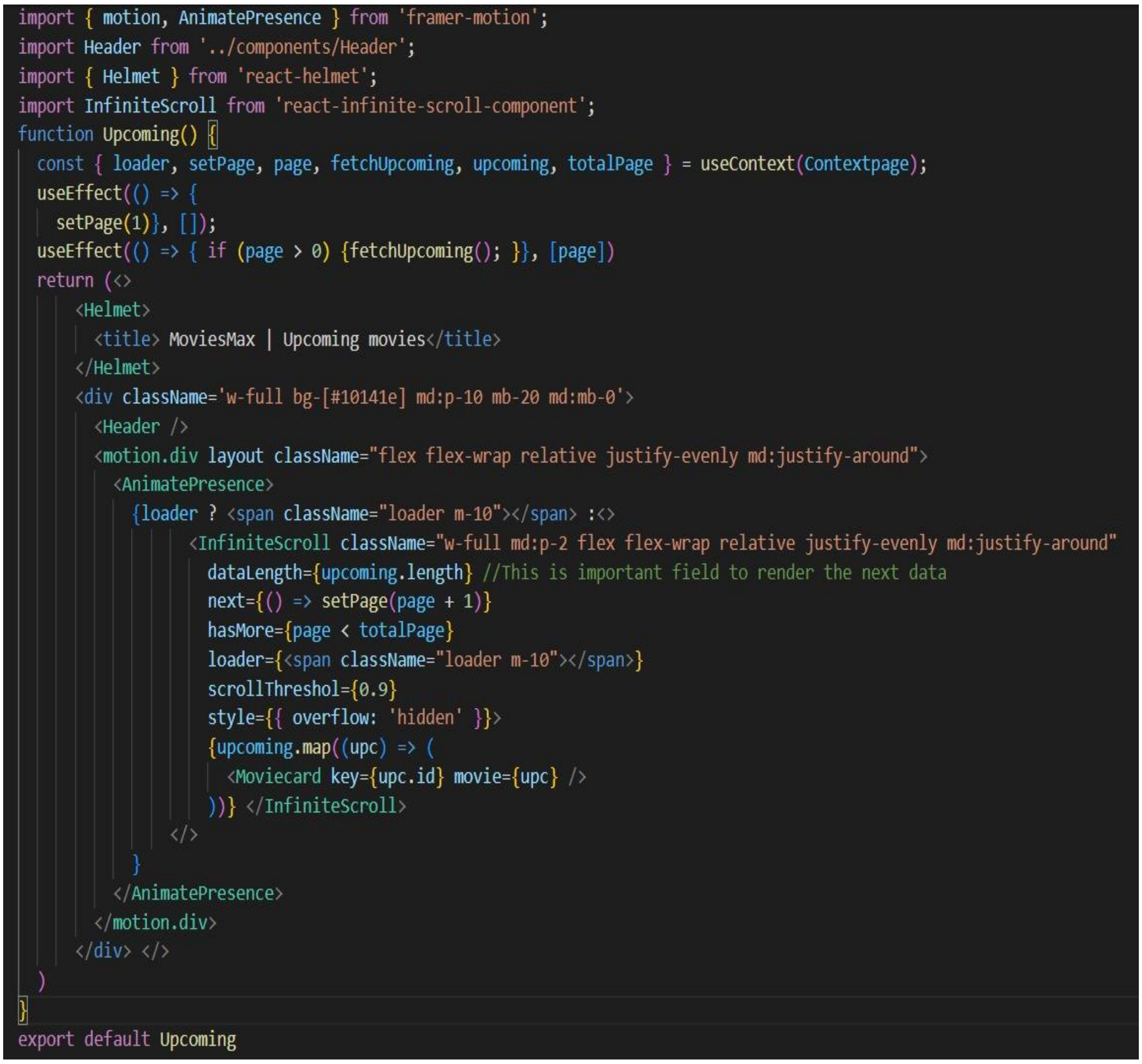
```jsx
import React, { useEffect, useContext } from 'react'
import Contextpage from '../Contextpage';
import Moviecard from '../components/Moviecard';
import { motion, AnimatePresence } from 'framer-motion';
import Header from '../components/Header';
import { Link, useParams } from 'react-router-dom'
import { HiChevronLeft } from "react-icons/hi";
function Search() {
    const { searchedMovies, loader, page, setPage, totalPage, setMovies, activegenre, filteredGenre, fetchSearch }
    = useContext(Contextpage);
    const { query } = useParams()
    useEffect(() => {
        fetchSearch(query);
    }, [query]); // Only re-run if 'query' or 'fetchSearch' changes
    return (
        <section>
            <Link to="/" className='fixed z-10 text-4xl text-black bg-white m-3 md:m-5 rounded-full'><HiChevronLeft /></Link>
            <div className='w-full bg-[#10141e] md:p-10 mb-20 md:mb-0'>
                <Header />
                <motion.div
                    layout
                    className="flex flex-wrap relative justify-evenly md:justify-around">
                    <AnimatePresence>
                        {
                            loader ? <span className="loader m-10"></span> :
                                <>
                                        {searchedMovies.map((movie) => (
                                            <Moviecard key={movie.id} movie={movie} />
                                        ))}
                                </>
                        }
                    </AnimatePresence>
                </motion.div>
            </div>
        </section> )
}
export default Search
```

Trending.jsx:



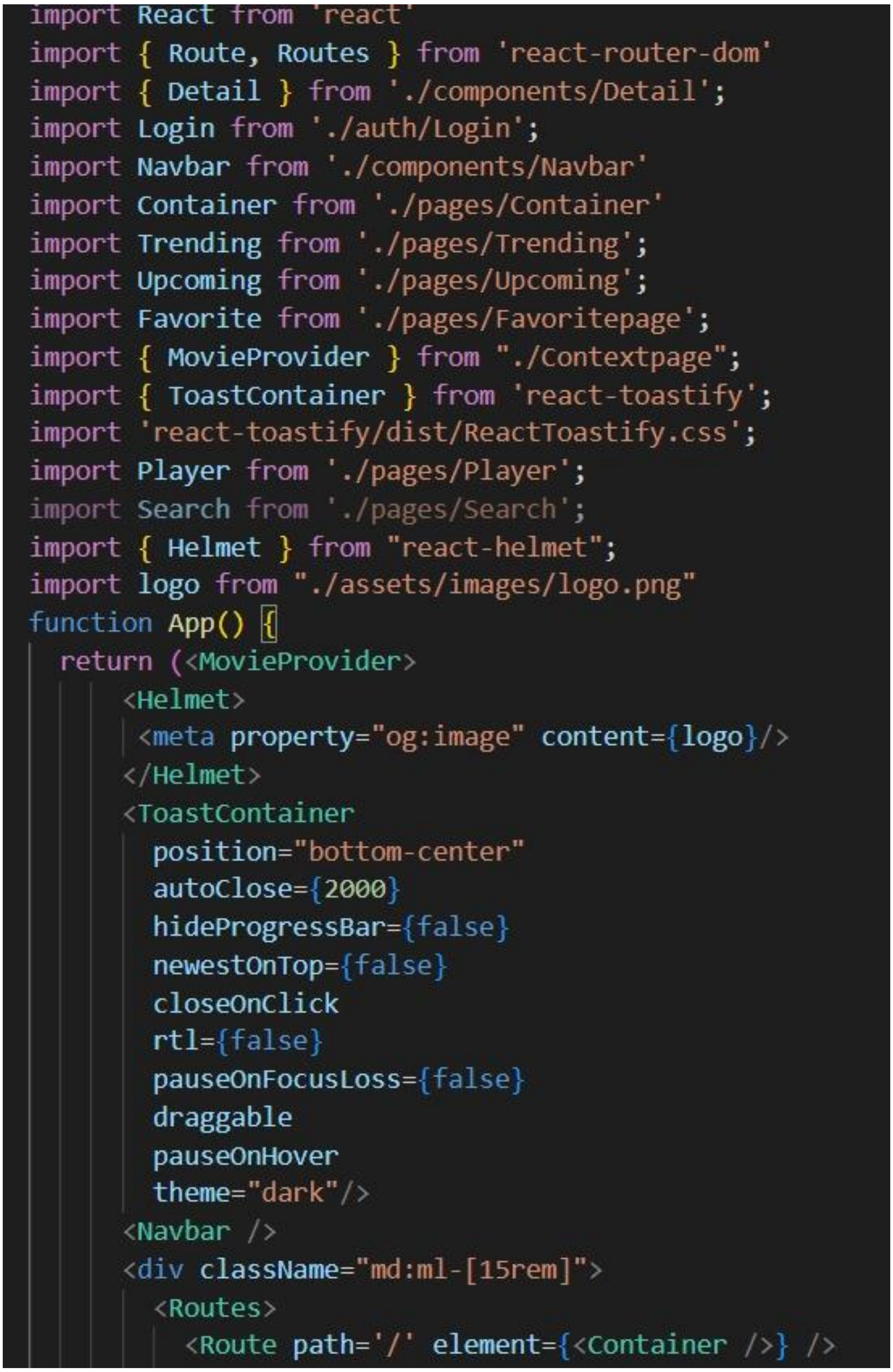
```
import React, { useEffect, useContext } from 'react'
import Contextpage from '../Contextpage';
import Moviecard from '../components/Moviecard';
import { motion, AnimatePresence } from 'framer-motion';
import Header from '../components/Header';
import { Helmet } from 'react-helmet';
import InfiniteScroll from 'react-infinite-scroll-component';
function Trending() {
    const { loader, page, setPage, fetchTrending, trending, totalPage } = useContext(Contextpage);
    useEffect(() => { setPage(1) }, []);
    useEffect(() => { if (page > 0) {fetchTrending(); } }, [page])
    return (  <>
                <Helmet>
                    <title>MovieMax A2Z Movies | Trending</title>
                </Helmet>
                <div className='w-full bg-[#10141e] md:p-10 mb-20 md:mb-0'>
                    <Header />
                    <motion.div layout className="flex flex-wrap relative justify-evenly md:justify-around">
                        <AnimatePresence>
                            { loader ? <span className="loader m-10"></span> :    <>
                          <InfiniteScroll
                           className="w-full md:p-2 flex flex-wrap relative justify-evenly md:justify-around"
                             dataLength={trending.length}   next={() => setPage(page + 1)}
                               hasMore={page < totalPage}
                             loader={<span className="loader m-10"></span>}
                               scrollThreshol={0.9}
                             style={{ overflow: 'hidden' }}      >
                             {trending.map((tred) => (
                             <Moviecard key={tred.id} movie={tred} />        ))}
                                </InfiniteScroll>        </>
                            }
                        </AnimatePresence> </motion.div>
                </div>
            </>  )
}
export default Trending
```

Upcoming.jsx:

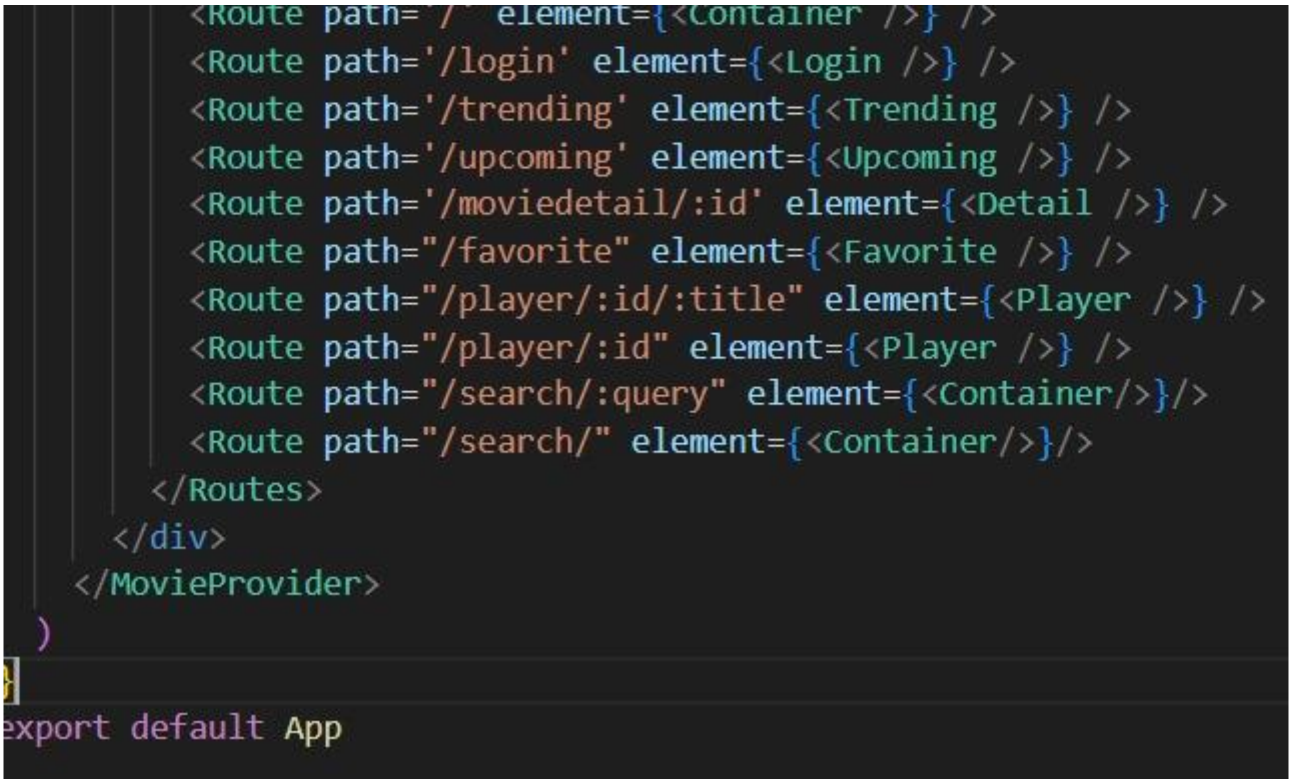


```
import { motion, AnimatePresence } from 'framer-motion';
import Header from '../components/Header';
import { Helmet } from 'react-helmet';
import InfiniteScroll from 'react-infinite-scroll-component';
function Upcoming() {
  const { loader, setPage, page, fetchUpcoming, upcoming, totalPage } = useContext(Contextpage);
  useEffect(() => {
    setPage(1)}, []);
  useEffect(() => { if (page > 0) {fetchUpcoming(); }}, [page])
  return (<>
      <Helmet>
        <title> MoviesMax | Upcoming movies</title>
      </Helmet>
      <div className='w-full bg-[#10141e] md:p-10 mb-20 md:mb-0'>
        <Header />
        <motion.div layout className="flex flex-wrap relative justify-evenly md:justify-around">
          <AnimatePresence>
            {loader ? <span className="loader m-10"></span> :<>
                  <InfiniteScroll className="w-full md:p-2 flex flex-wrap relative justify-evenly md:justify-around"
                    dataLength={upcoming.length} //This is important field to render the next data
                    next={() => setPage(page + 1)}
                    hasMore={page < totalPage}
                    loader={<span className="loader m-10"></span>}
                    scrollThreshol={0.9}
                    style={{ overflow: 'hidden' }}>
                    {upcoming.map((upc) => (
                      <Moviecard key={upc.id} movie={upc} />
                    ))} </InfiniteScroll>
                </>
            }
          </AnimatePresence>
        </motion.div>
      </div> </>
  )
}
export default Upcoming
```

app.jsx:

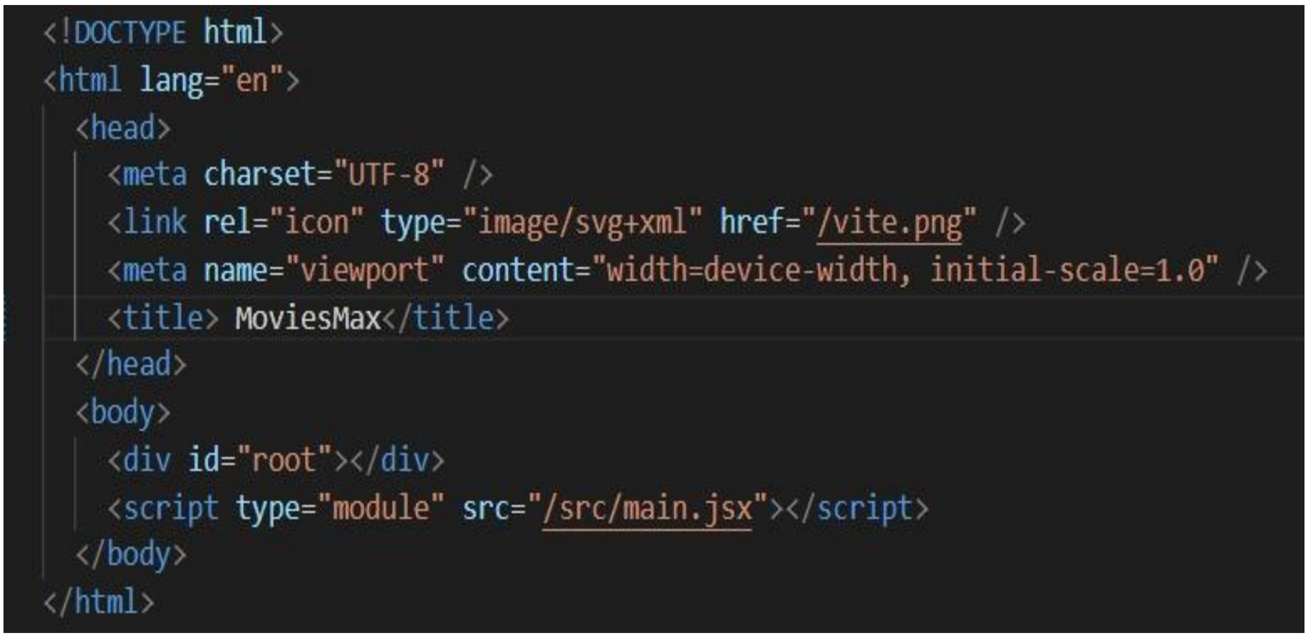


```
import React from 'react'
import { Route, Routes } from 'react-router-dom'
import { Detail } from './components/Detail';
import Login from './auth/Login';
import Navbar from './components/Navbar'
import Container from './pages/Container'
import Trending from './pages/Trending';
import Upcoming from './pages/Upcoming';
import Favorite from './pages/Favoritepage';
import { MovieProvider } from "./Contextpage";
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import Player from './pages/Player';
import Search from './pages/Search';
import { Helmet } from "react-helmet";
import logo from "./assets/images/logo.png"
function App() {
  return (<MovieProvider>
      <Helmet>
       <meta property="og:image" content={logo}/>
      </Helmet>
      <ToastContainer
        position="bottom-center"
        autoClose={2000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
        pauseOnFocusLoss={false}
        draggable
        pauseOnHover
        theme="dark"/>
      <Navbar />
      <div className="md:ml-[15rem]">
        <Routes>
          <Route path='/' element={<Container />} />
```



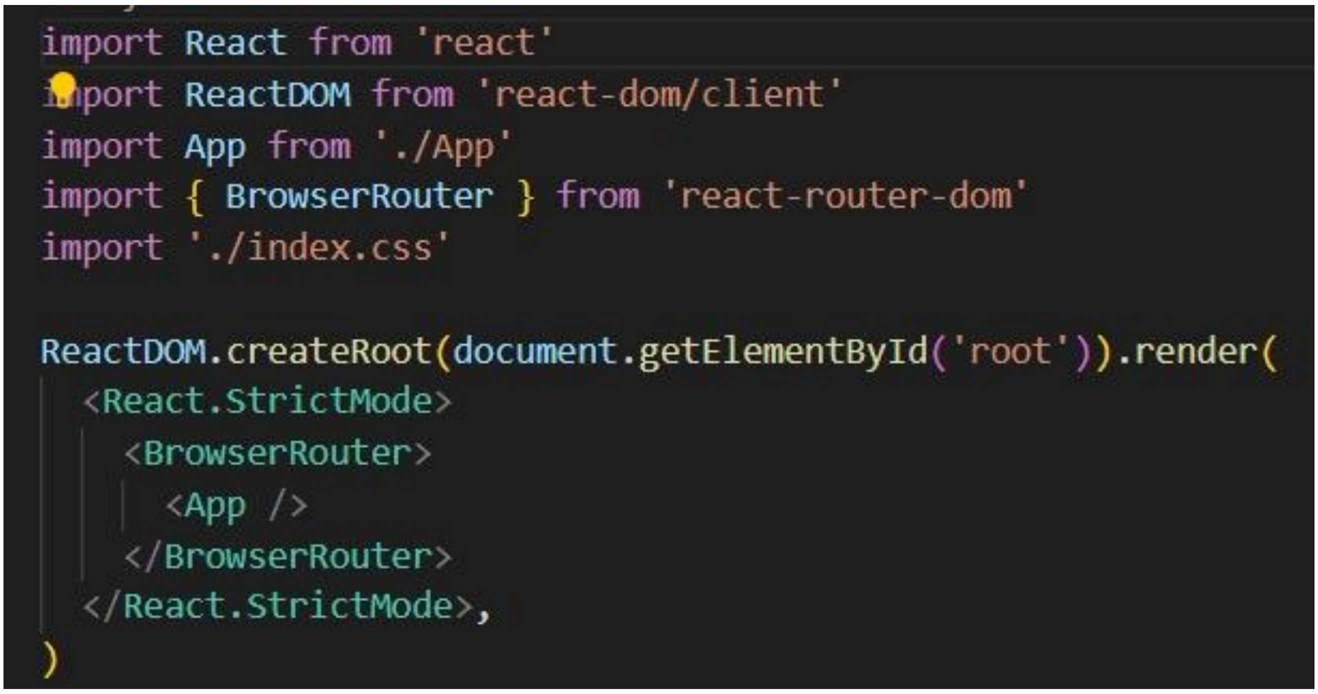
```
          <Route path='/' element={<Container />} />
          <Route path='/login' element={<Login />} />
          <Route path='/trending' element={<Trending />} />
          <Route path='/upcoming' element={<Upcoming />} />
          <Route path='/moviedetail/:id' element={<Detail />} />
          <Route path="/favorite" element={<Favorite />} />
          <Route path="/player/:id/:title" element={<Player />} />
          <Route path="/player/:id" element={<Player />} />
          <Route path="/search/:query" element={<Container/>}/>
          <Route path="/search/" element={<Container/>}/>
        </Routes>
      </div>
    </MovieProvider>
  )
}
xport default App
```

Index.html:



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title> MoviesMax</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

main.jsx:



```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import { BrowserRouter } from 'react-router-dom'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
)
```

5.3 UI Screenshots

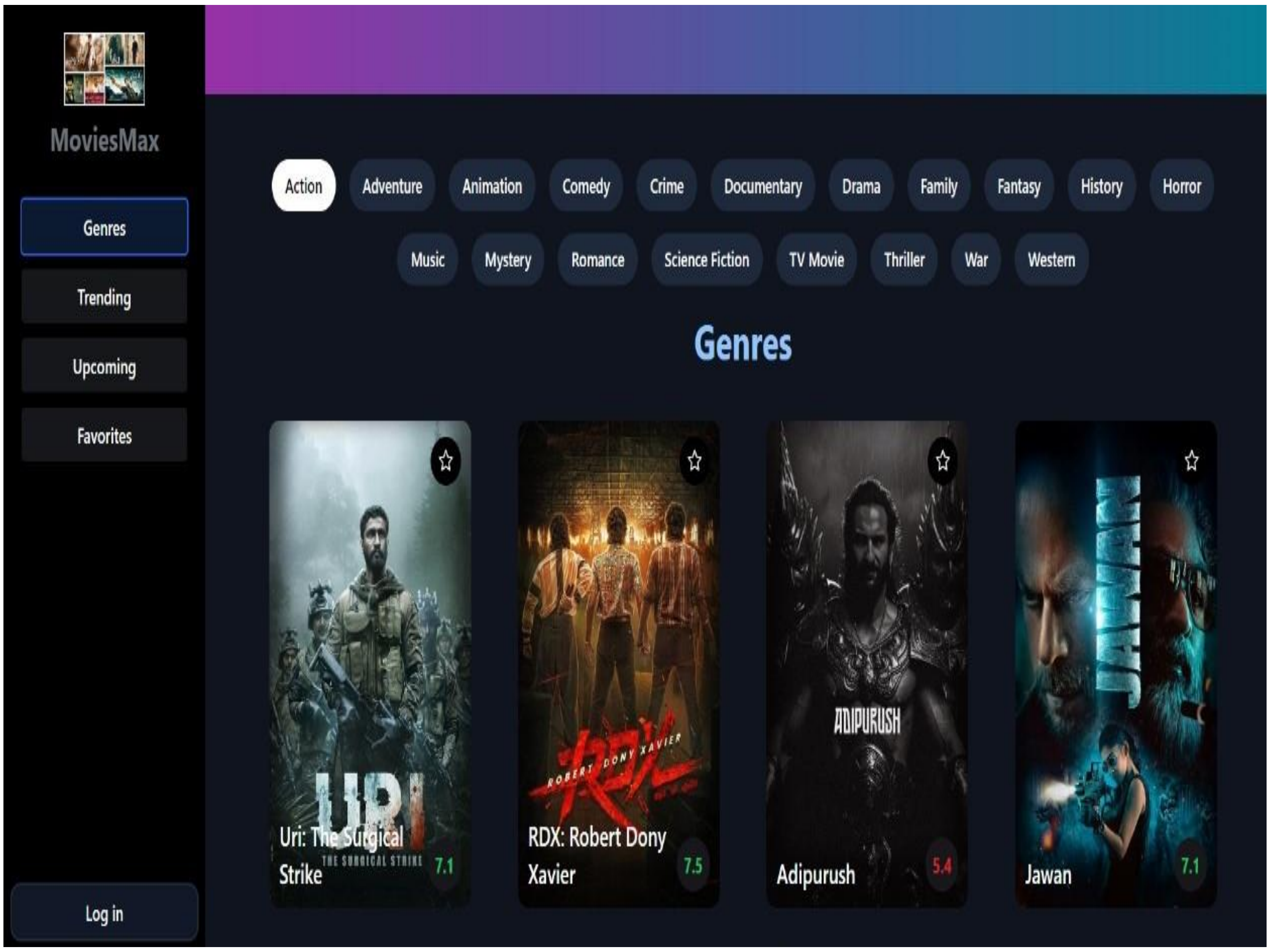


Fig. 5.3.1 – Genres

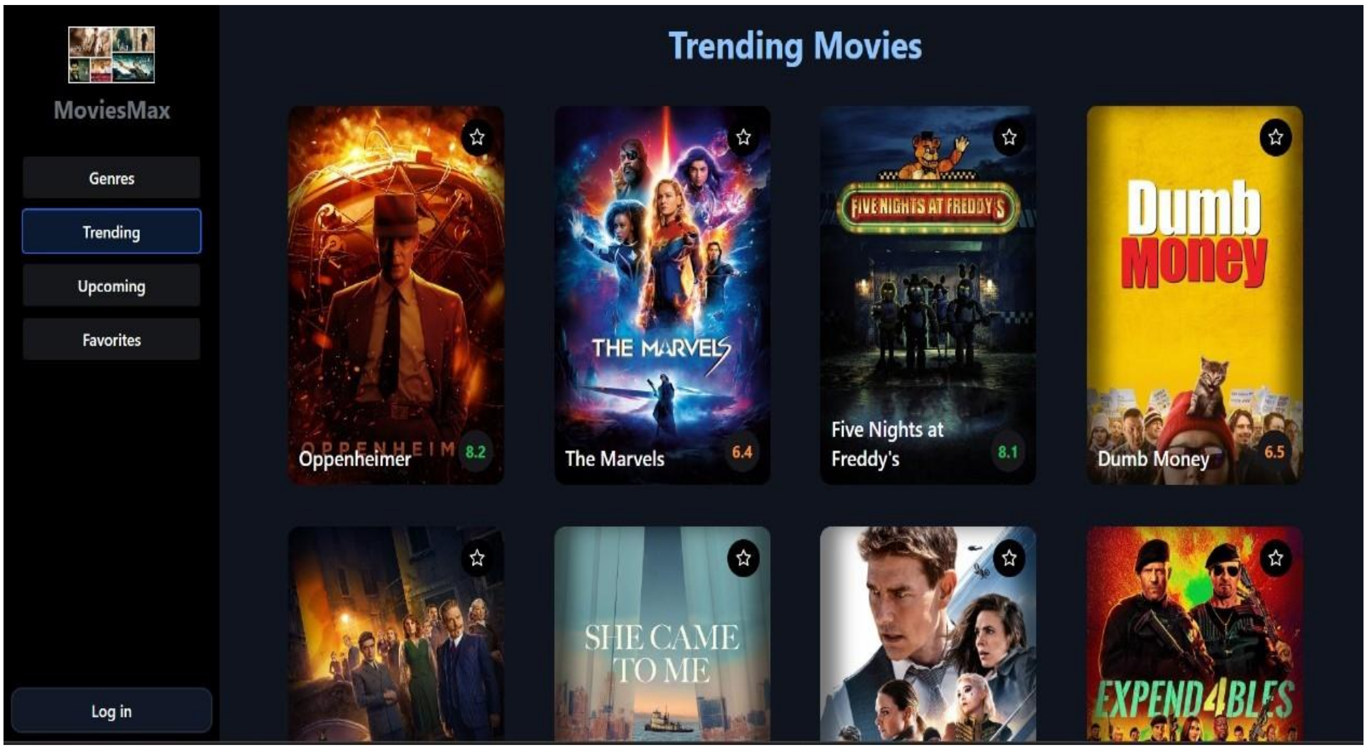


Fig. 5.3.2 – Trending

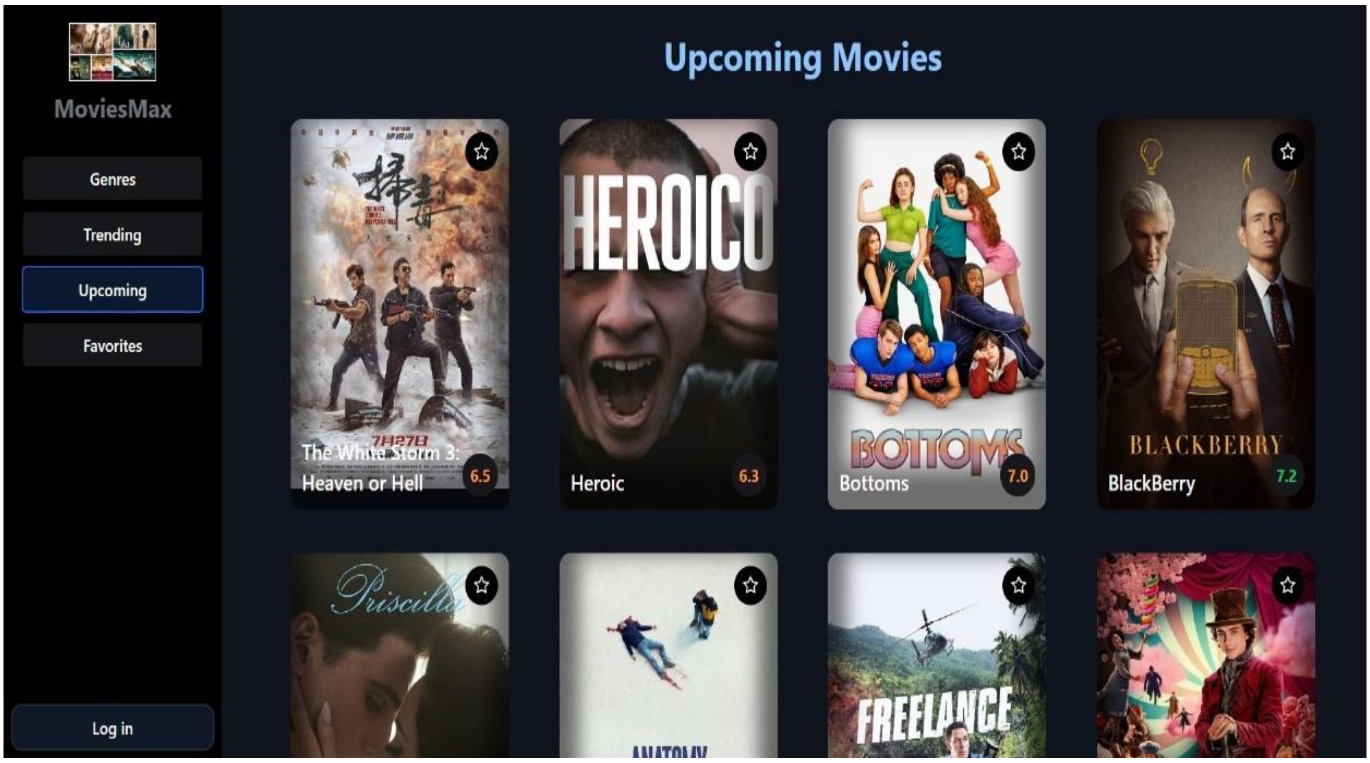
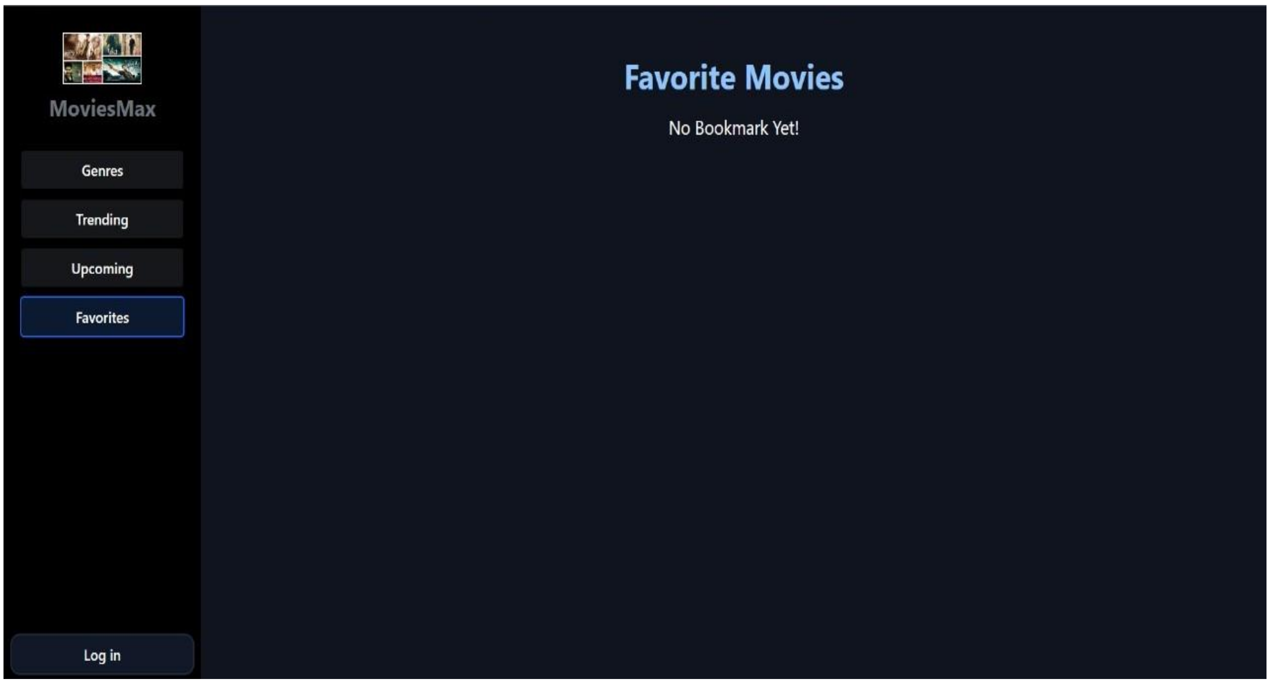


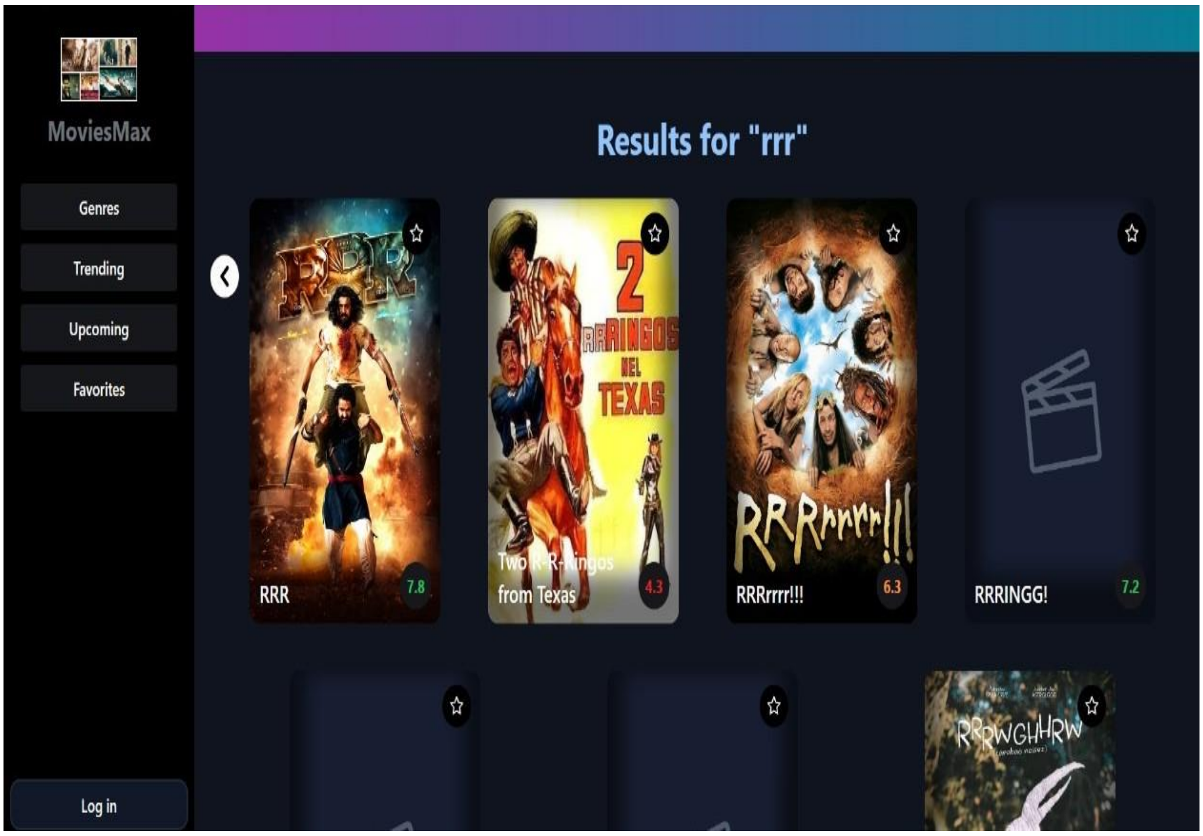
Fig. 5.3.3 – Upcoming



Favorite Movies

No Bookmark Yet!

Fig. 5.3.4 – Favourites



Results for "rrr"

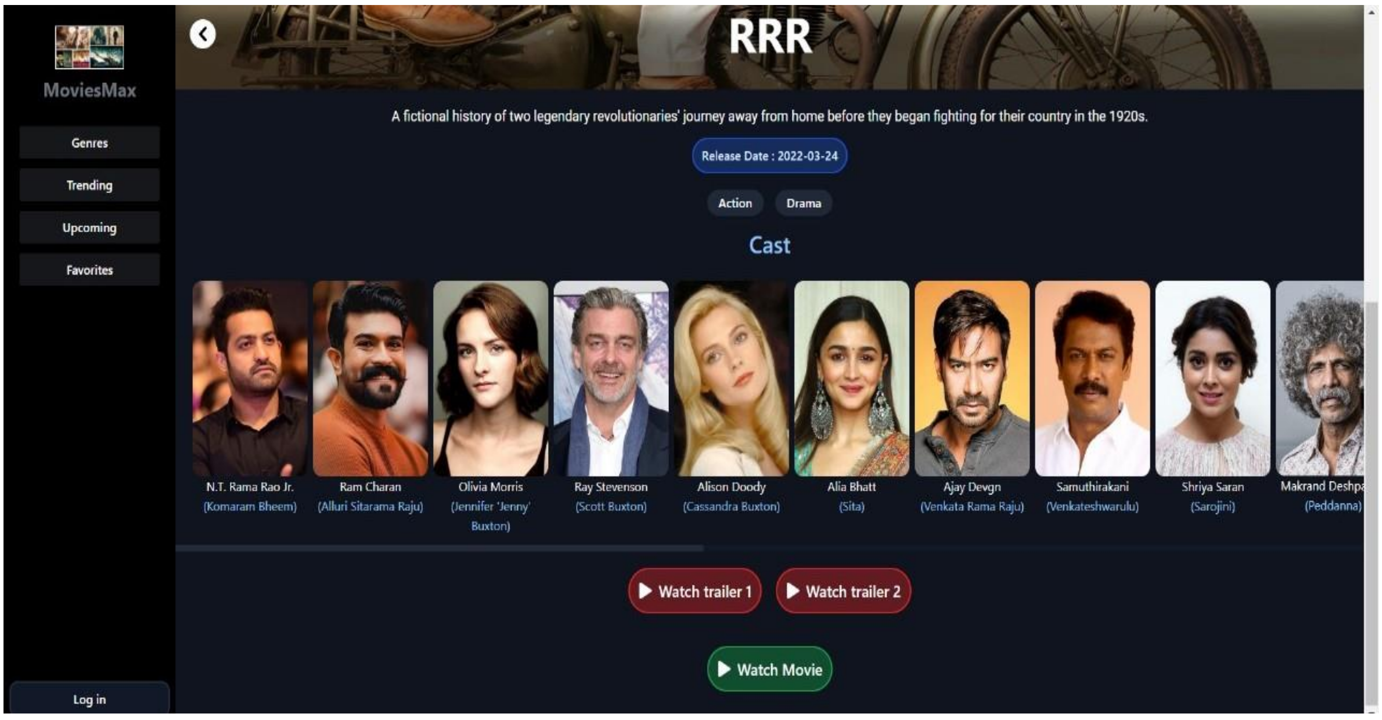
RRR 7.8

Two R-R-Ringos from Texas 4.3

RRRrrrr!!! 6.3

RRRINGG! 7.2

Fig. 5.3.5 – Search for a movie



RRR

A fictional history of two legendary revolutionaries' journey away from home before they began fighting for their country in the 1920s.

Release Date : 2022-03-24

Action Drama

Cast

N.T. Rama Rao Jr. (Komaram Bheem)
Ram Charan (Alluri Sitarama Raju)
Olivia Morris (Jennifer 'Jenny' Buxton)
Ray Stevenson (Scott Buxton)
Alison Doody (Cassandra Buxton)
Alia Bhatt (Sita)
Ajay Devgn (Venkata Rama Raju)
Samuthirakani (Venkateshwarulu)
Shriya Saran (Sarojini)
Makrand Deshpa (Peddanna)

Watch trailer 1 Watch trailer 2

Watch Movie

Fig. 5.3.6 – Movie Info

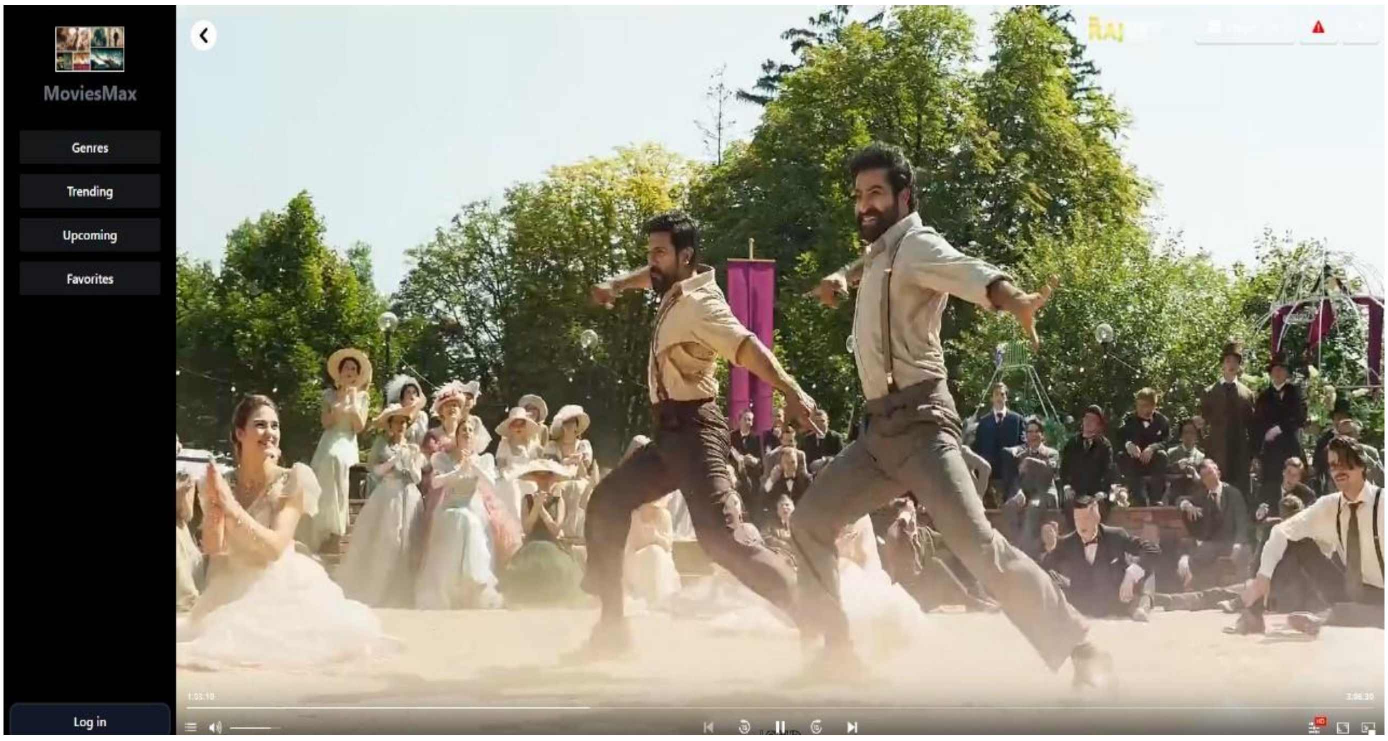


Fig. 5.3.7 – Streaming a movie

CHAPTER – 6

6. TESTING

6.1 Introduction

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as

- A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

Who does Testing do?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities:

1. Software Tester

2. Software Developer

3. Project Lead/Manager

4. End User

Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are:

1. Functional Testing
2. Non-Functional Testing

Functional Testing

This is a type of testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Non-Functional Testing

It is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

Software Testing Life Cycle

The process of testing software in a well-planned and systematic way is known as software testing lifecycle (STLC).

Different organizations have different phases in STLC however generic Software Test Life Cycle (STLC) for waterfall development model consists of the following phases:

1. Requirements Analysis
2. Test Planning
3. Test Analysis
4. Test Design

Requirements Analysis

In this phase testers analyse the customer requirements and work with developers during the design phase to see which requirements are testable and how they are going to test those requirements. It is very important to start testing activities from the requirements phase itself because the cost of fixing defect is very less if it is found in requirements phase rather than in future phases.

Test Planning

In this phase all the planning about testing is done like what needs to be tested, how the testing will be done, test strategy to be followed, what will be the test environment, what test methodologies will be followed, hardware and software availability, resources, risks etc.

Test Analysis

After test planning phase is over test analysis phase starts, in this phase we need to dig deeper into project and figure out what testing needs to be carried out in each SDLC phase. Automation activities are also decided in this phase, if automation needs to be done for software product, how will the automation be done, how much time will it take to automate and which features need to be automated. Non-functional testing areas (Stress and performance testing) are also analysed and defined in this phase.

Test Design

In this phase various black-box and white-box test design techniques are used to design the test cases for testing, testers start writing test cases by following those design techniques, if automation testing needs to be done then automation scripts also need to written in this phase.

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification

1. SOFTWARE VALIDATION

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

Validation ensures the product under development is as per the user requirements. Validation emphasizes user requirements.

2. SOFTWARE VERIFICATION

Verification is the process of confirming if the software is meeting the business requirements and is developed adhering to the proper specifications and methodologies. Verification ensures the product being developed is according to design specifications. Verification answers the question– "Are we developing this product by firmly following Verification concentrate on the design and system specifications.

3. TARGET OF THE TEST ARE

• Errors -These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, considered as an error.

• Fault - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.

• Failure - failure is said to be the inability of the system to perform the desired t ask. Failure occurs when fault exists in the system.

6.1.1 Test objectives

1. User Authentication:
   - Objective: Ensure a smooth user authentication process, allowing users to log in if authorized and prompting registration for new users.
   - Tests:
     - Verify that registered users can log in successfully.
     - Confirm that unauthorized users are redirected to the registration page.

2. Movie Search and Bookmarking:
   - Objective: Validate the movie search functionality and the ability to bookmark favorite movies.
   - Tests:
     - Search for movies by title and confirm accurate search results.
     - Verify that users can bookmark their favorite movies for later viewing.

3. Genre Sorting and Filtering:
   - Objective: Confirm the system's capability to sort and filter movies based on genres.
   - Tests:
     - Sort movies by genre and verify the correct grouping of movies.
     - Apply genre filters and ensure only relevant movies are displayed.

4. Movie Details and Trailer Playback:
   - Objective: Validate the presentation of detailed information for each movie and the ability to watch trailers.
   - Tests:
     - Navigate to a movie's details page and confirm the accuracy of displayed information.
     - Play trailers for selected movies and verify seamless playback.

5. User Interaction and Navigation:
   - Objective: Confirm smooth user interaction and navigation within the application.
   - Tests:
     - Test the responsiveness of buttons and features, ensuring they perform as expected.
     - Verify seamless navigation between different sections of the application.

6. Upcoming and Trending Movies:
   - Objective: Ensure accurate display of upcoming and trending movies.
   - Tests:
     - Confirm that the upcoming movies section accurately reflects films set to be released soon.
     - Verify that the trending movies section displays the latest popular releases.
7. Performance Under Load:
   - Objective: Evaluate system performance under varying user loads.
   - Tests:

- Conduct load testing to ensure the application remains responsive with multiple concurrent users.
- Assess how performance metrics are impacted during peak usage.

8. Error Handling and User Feedback:
- Objective: Ensure the application gracefully handles errors and provides clear user feedback.
- Tests:
    - Intentionally trigger errors (e.g., invalid search queries) and verify the system's response.

UNIT	PURPOSE	STATUS
Search Movies	This unit tests the search functionality with an invalid query.	Success
Bookmark Movie	This unit tests the ability to bookmark a movie for later viewing.	Success
Bookmark Movie	This unit tests the attempt to bookmark a movie that does not exist.	Failure
Genre Sorting	This unit tests the sorting of movies based on genre.	Success
Genre Sorting	This unit tests the sorting with an invalid genre.	Success
View Movie Details	This unit tests the display of detailed information for a selected movie.	Success
View Movie Details	This unit tests accessing details for a non-existent movie.	Failure
Watch Trailer	This unit tests the ability to watch trailers for selected movies.	Success
Watch Trailer	This unit tests attempting to watch a trailer for a movie without a trailer.	Success
Upcoming Movies	This unit tests the accurate display of upcoming movies.	Success
Upcoming Movies	This unit tests upcoming movies display with invalid data.	Failure
Trending Movies	This unit tests the display of trending movies.	Success
Trending Movies	This unit tests trending movies display with invalid data.	Failure
User Authentication	This unit tests the login of a registered user.	Success
User Authentication	This unit tests an invalid login attempt.	Success
User Registration	This unit tests successful user registration.	Success
User Registration	This unit tests unsuccessful user registration with invalid details.	Success
Add Movie Rating	This unit tests the addition of a movie rating by a user.	Success
Add Movie Rating	This unit tests attempting to rate a movie without proper authentication.	Success

Table 6.1.1 Test Objectives

6.1.2 Test Strategies

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	identified classes of valid input must be accepted.
Invalid Input	identified classes of invalid input must be rejected.
Functions	identified functions must be exercised.
Output	identified classes of application outputs must be exercised.
Systems/Procedures	interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g., components in a software system or - one step up - software applications at the company level - interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results All the test cases mentioned above passed successfully. No defects encountered.

6.2 Test Cases

Module	Action	Input	Output
Movie Bookmarking	Add	Movie ID: "M1"	Added into user's bookmarked list
Movie Bookmarking	Remove	Movie ID: "M2"	Removed from user's bookmarked list
Movie Search	Search	Title: "The Matrix"	Search results include "The Matrix"
Movie Sorting	Sort by Genre	Genre: "Action"	Movies correctly grouped under "Action"
View Movie Details	Click on Movie	Movie ID: "M1"	Detailed information about "M1" displayed
Watch Trailer	Play Trailer	Movie ID: "M1"	Trailer for "M1" plays successfully
Upcoming Movies	View Section	-	Upcoming movies section displays accurately
Trending Movies	View Section	-	Trending movies section displays accurately
User Authentication	Login	Credentials: User1	User1 logs in successfully

Table 6.4.2. Test Cases

CONCLUSION

In conclusion, the React movie app represents the culmination of our efforts to create an engaging and user-friendly platform for movie enthusiasts. Through a combination of modern web technologies, including ReactJS, the TMDB API, Firebase Google Authentication, and Framer Motion, we have successfully developed a feature-rich movie website that caters to the diverse needs and preferences of movie lovers.

This project began with a clear vision: to provide users with a central hub for discovering, exploring, and enjoying the world of cinema. Our app allows users to search for movies by title, sort them by genre, view trending and upcoming releases, and even bookmark their favorite films for future viewing. The integration of Google Authentication enhances the user experience by providing a seamless and secure login process.

The motivation behind this project stemmed from our desire to create a platform that not only simplifies the movie discovery process but also fosters a sense of community among movie enthusiasts. We aimed to address the ever-growing demand for personalized movie recommendations, reliable information, and a space for sharing one's passion for film.

The successful execution of this project not only met but exceeded our initial goals. The app offers a visually appealing and responsive user interface, ensuring that users can effortlessly navigate its features and access a vast database of movies. The addition of Framer Motion animations adds a touch of elegance to the overall user experience.

As we reflect on the journey of building this React movie app, it is clear that our work is far from complete. The future holds numerous opportunities for enhancements and expansions, including the addition of user-generated content, advanced recommendation algorithms, and interactive features that can transform the app into a vibrant movie community.

In summary, the React movie app is a testament to our dedication and passion for the world of cinema. We are proud of what we have accomplished, and we look forward to further developing and improving this platform to continue serving the needs and interests of movie enthusiasts around the world.

FUTURE ENHANCEMENTS

1. User Reviews and Ratings:

Allow users to leave reviews and ratings for movies, creating a more interactive community and helping others make informed viewing choices.

2. User-Generated Content: Enable users to contribute by adding movie recommendations, creating watchlists, and sharing their custom movie lists with others.

3. Notifications: Implement a notification system to alert users about upcoming movies in their preferred genres, new releases, or when movies on their watchlist become available.

4. User Profiles: Enhance user profiles with additional information, such as a list of favorite actors, directors, or specific movie preferences, to improve movie recommendations.

5. Social Integration: Integrate social media sharing options, allowing users to share their movie discoveries, reviews, and watchlists with their friends and followers.

6. Advanced Search Filters: Implement advanced search filters, such as filtering by release year, actor, director, or awards, to help users find movies more precisely.

7. Multi-Language Support: Extend language support to cater to a global audience, providing content in various languages and allowing users to switch between them.

8. Bookmark Folders: Allow users to organize their bookmarked movies into folders or categories for easier management.

9. Personalized Content Notifications: Send personalized email or in-app notifications to users based on their activity, preferences, and movie history.

10. Offline Viewing: Develop an offline viewing feature that enables users to download movies for later offline viewing.

REFERENCES

1. ReactJS: Official React Documentation.

   Retrieved from https://reactjs.org/

2. TMDB API:

   The Movie Database (TMDB) API Documentation. Retrieved from https://developers.themoviedb.org/3/getting-started/introduction

3. Firebase Authentication:

   Firebase Authentication Documentation. Retrieved from https://firebase.google.com/docs/auth

4. Framer Motion: Framer Motion Documentation.

   Retrieved from https://www.framer.com/api/motion/

5. React-Router: React-Router Documentation.

   Retrieved from https://reactrouter.com/web/guides/quick-start

6. LazyLoadImage:

   React Lazy Load Image Component Documentation. Retrieved from https://www.npmjs.com/package/react-lazy-load-image-component

7. Slugify:

   React Slugify Documentation. Retrieved from https://www.npmjs.com/package/react-slugify

8. The Movie Database (TMDB):

   Official TMDB Website. Retrieved from https://www.themoviedb.org/

9. Google Authentication:

   Google Firebase Authentication Guide. Retrieved from https://firebase.google.com/products/auth

10. Material-UI: Material-UI React Component Library.
    Retrieved from https://mui.com/

BIBLIOGRAPHY

1. ReactJS Documentation:
   - Facebook. "React - A JavaScript library for building user interfaces."
   - [https://reactjs.org/docs/getting-started.html] (https://reactjs.org/docs/getting-started.html)

2. TMDB API Documentation:
   - The Movie Database. "TMDB API Documentation."
   - https://developers.themoviedb.org/3/getting-started/introduction

3. Firebase Documentation:
   - Google. "Firebase Documentation."
   - https://firebase.google.com/docs

4. Framer Motion Documentation:
   - Framer. "Framer Motion Documentation."
   - https://www.framer.com/docs/

5. Modern Authentication with Google:
   - Google Identity Platform. "Modern Authentication with Google."
   - https://developers.google.com/identity