Project Report

On

MovieMagic-A Movie Matchmaking System

Submitted in the Partial Fulfillment of the Requirement for the award of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY (2024)

By

SIDDHARTHA MISHRA (2001220130116)

SUJAL VARSHNEY (2001220130117)

Under the Guidance

Of

Dr. Vibha Srivastava

SHRI RAMSWAROOP MEMORIAL COLLEGE OF ENGINEERING AND MANAGEMENT, LUCKNOW

Affiliated to

Dr. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY,

UTTAR PRADESH, LUCKNOW

DEPARTMENT OF INFORMATION TECHNOLOGY

SHRI RAMSWAROOP MEMORIAL COLLEGE OF ENGINEERING AND MANAGEMENT

CERTIFICATE

Certified that the project entitled **"MovieMagic- A Movie Matchmaking System"** submitted by SIDDHARTHA MISHRA, [Roll No. 2001220130116] and SUJAL VARSHNEY, [RollNo. 2001220130117] in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Information Technology) of Dr. APJ Abdul KalamTechnical University (Uttar Pradesh, Lucknow), is a record of students' own work carried under our supervision and guidance. The project report embodies results of original work and studies carried out by students and the contents do not forms the basisfor the award of any other degree to the candidate or to anybody else.

Dr. Vibha Srivastava — Project Guide

Prof. Ajay Kumar Srivastava — Head Of Department

DEPARTMENT OF INFORMATION TECHNOLOGY

SHRI RAMSWAROOP MEMORIAL COLLEGE OF ENGINEERING AND MANAGEMENT

DECLARATION

We hereby declare that the project entitled **"MovieMagic- A Movie Matchmaking System"** submitted by us in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Information Technology) of Dr. APJ Abdul Kalam Technical University, is a record of our own work carried under the supervision and guidance of **Dr. Vibha Srivastava (Asst. Professor).**

To the best of our knowledge this project has not been submitted to Dr. APJ Abdul Kalam Technical University or any other University or Institute for the award of any degree.

SIDDHARTHA MISHRA **SUJAL VARSHNEY**

[2001220130116] **[2001220130117]**

DEPARTMENT OF INFORMATION TECHNOLOGY

SHRI RAMSWAROOP MEMORIAL COLLEGE OF ENGINEERING AND MANAGEMENT

ACKNOWLEDGEMENT

We extend our heartfelt gratitude and sincere thanks to our project guide **Dr. Vibha Srivastava** and Head Of Department **Prof. Ajay Kr. Srivastava** for their continuous guidance, support and motivation during this project. We are thankful to them for their constant constructive and valuable suggestion which benefited us a lot in developing this project. We take this opportunity to express our heartfelt thanks to the **Information Technology Department** and **Shri Ramswaroop Memorial College Of Engineering and Management** for the available relevant resources as well as the invigorating study atmosphere they have provided.

We would also like to express our special gratitude to the all the staff members and faculty of IT Department for their efforts dedicated to us and provided helpful feedback.

SIDDHARTHA MISHRA **[2001220130116]**

SUJAL VASRHNEY **[2001220130117]**

PREFACE

This project report delineates the conception and realization of a cutting-edge movie recommendation system application, designed to operate seamlessly across both iOS and Android platforms using React Native. At its core, the system integrates a bespoke machine learning model tailored for content-based movie recommendations, leveraging a diverse dataset encompassing over 5000 movies. The model undergoes meticulous preprocessing and tagging, followed by vectorization utilizing the bag-of-words model. Recommendation generation is facilitated through cosine similarity calculations, ensuring the delivery of personalized movie suggestions tailored to individual user preferences.

The backend infrastructure, powered by the Flask framework, plays a pivotal role in deploying the machine learning model via an API, seamlessly integrating with the frontend for cohesive user interaction. Firebase authentication streamlines user management, offering a user-friendly login process while accommodating new user registrations with ease.

Upon successful authentication, users are greeted with a dynamic home screen showcasing trending and top-rated movies, augmented by a powerful search feature for comprehensive movie exploration. Furthermore, the application boasts an interactive chat feature driven by the OpenAI GPT API, enriching user engagement by providing instant responses to movie-related queries, enhancing the overall user experience.

Detailed movie screens offer an immersive experience, presenting comprehensive information including cast details sourced from the TMDB API. Additionally, users are provided with similar movie recommendations generated by the machine learning model, fostering further exploration and within the cinematic realm.

This project represents the convergence of advanced technologies to deliver a sophisticated and tailored movie recommendation experience, redefining the paradigm of personalized content discovery in the realm of entertainment. Through meticulous implementation and integration, the application offers users a seamless journey into the world of cinema and entertainment, curated to their unique tastes and preferences.

ABSTRACT

This project abstract encapsulates the development of a cross-platform movie recommendation application using React Native, bolstered by a backend infrastructure featuring a custom machine learning model. With a dataset exceeding 5000 movies, the system employs advanced techniques such as preprocessing, tagging, and vectorization using the bag-of-words model to generate personalized movie recommendations. Cosine similarity calculations further refine the recommendations, ensuring relevance to individual user preferences. The Flask framework facilitates backend deployment, seamlessly integrating the machine learning model via an API with the frontend. Firebase authentication streamlines user management, offering a user-friendly login process and accommodating new user registrations.

Upon authentication, users gain access to a dynamic home screen displaying trending and top-rated movies, supplemented by a robust search feature for comprehensive movie exploration. An interactive chat feature powered by the OpenAI GPT API enhances user engagement by providing real-time responses to movie-related queries. Detailed movie screens offer immersive experiences, presenting comprehensive information and similar movie recommendations generated by the machine learning model. This project embodies the fusion of advanced technologies to deliver a tailored and immersive movie discovery experience, redefining personalized content exploration in the entertainment landscape.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In the digital age, the consumption of movies and media has significantly evolved, driven by advancements in technology and the widespread availability of online streaming services. As the volume of available content grows exponentially, finding the right movie that aligns with an individual's tastes and preferences has become increasingly challenging. This project addresses this challenge by developing a movie recommendation system application, leveraging the capabilities of modern technologies to provide personalized movie suggestions.

Our application is built using React Native, ensuring it is responsive and functional across both iOS and Android platforms. The backend infrastructure features a custom machine learning (ML) model designed to deliver accurate and relevant movie recommendations. This model is based on a Content-Based Filtering approach, utilizing a dataset of over 5000 movies. The process involves preprocessing and tagging movie data, followed by vectorization using the bag-of-words model. Similarities between movies are calculated through cosine similarity, enabling the system to recommend movies with similar characteristics to the user’s preferences.

User authentication and management are handled by Firebase, providing a secure and efficient login and signup process. Once authenticated, users are presented with a home screen showcasing trending and top-rated movies. They can also search for specific movies and interact with a chatbot powered by the OpenAI GPT API, which answers all movie-related queries. Additionally, each movie detail screen provides comprehensive information retrieved from the TMDB API, including cast details and similar movie recommendations generated by our ML model.

This application not only simplifies the movie selection process but also enhances user engagement by offering a personalized and interactive movie browsing experience. Through this project, we aim to bridge the gap between users and the vast array of available cinematic content, making movie discovery both enjoyable and efficient.

CHAPTER 2

LITREATURE SURVEY

LITERATURE REVIEW:

Movie recommendation systems have garnered significant attention in recent years due to the exponential growth of digital streaming platforms and the increasing demand for personalized content discovery experiences. Content-based recommendation systems, like the one proposed in this project, have emerged as a prominent approach for addressing the challenge of recommending movies to users based on their individual preferences and viewing history.

A foundational aspect of content-based recommendation systems is the utilization of machine learning algorithms to analyze movie attributes and user preferences. Techniques such as natural language processing (NLP) for text analysis, vectorization methods like the bag-of-words model, and similarity measures such as cosine similarity have been extensively employed to compute the similarity between movies and generate recommendations.

Flask, a lightweight and versatile web framework, has gained popularity for deploying machine learning models via APIs, offering scalability and flexibility in integrating backend systems with frontend applications. This integration facilitates the seamless communication between the machine learning model and the React Native frontend, enabling efficient recommendation delivery to users.

Firebase has emerged as a robust solution for user authentication and management in mobile applications, offering developers a suite of tools for streamlined user registration, login, and session management. Its integration into the project ensures a seamless user experience, enhancing user engagement and retention.

Additionally, the incorporation of external APIs such as the TMDB API for accessing movie details and the OpenAI GPT API for the chat feature further enriches the application's functionality and user interaction.

Overall, existing literature underscores the importance of leveraging advanced technologies and methodologies, including machine learning, web frameworks, and APIs, to develop effective and engaging movie recommendation systems that cater to the diverse preferences of modern-day audiences. This project builds upon and extends these principles to create a comprehensive and immersive movie discovery platform tailored to individual user tastes and preferences.

CHAPTER 3

PROPSOED METHODOLOGY

3.1 PROBLEM DEFINITION:

The rapid proliferation of digital streaming platforms has led to an overwhelming abundance of movie choices for users, making it increasingly challenging to discover content aligned with their preferences. Existing movie recommendation systems often lack the granularity and personalization needed to cater to individual user tastes, resulting in suboptimal user experiences and reduced engagement.

To address this issue, there is a pressing need for a sophisticated movie recommendation system that can leverage advanced machine learning techniques to analyze user preferences and provide personalized movie recommendations. Additionally, there is a demand for a seamless and intuitive user interface that can efficiently deliver these recommendations across both iOS and Android platforms.

This project aims to develop a cross-platform movie recommendation application using React Native, supported by a backend infrastructure featuring a custom machine learning model. By harnessing the power of machine learning algorithms and integrating with popular web frameworks and APIs, the goal is to create a comprehensive solution that enhances user satisfaction and engagement in the movie discovery process.

3.2 AIMS AND OBJECTIVES

1. Create and implement user-friendly interfaces for mobile applications using React Native.
2. Integrate with TMDb API to access and store video data including genre, rating, card, and group.
3. Develop technologies such as collaborative filtering or content-based filtering to determine user preferences and video content.
4. Let users interact with features like commenting on videos, saving to favorites, and getting recommendations.
5. Conduct extensive testing to ensure app functionality, performance and usability across different devices and platforms.

3.3 SOLUTION APPROACH

The Iterative Waterfall Model is a software development method that blends the sequential steps of the traditional Waterfall Model with the flexibility of iterative design. It allows for improvements and changes to be made at each stage of the development process, instead of waiting until the end of the project. The iterative waterfall model offers feedback paths from every phase to its previous stages, which is the main change from the classical waterfall model. The feedback paths allow the phase to be reworked in which mistakes are made and these changes are mirrored in the later stages. In this model, the software development process is split into a number of iterations, or cycles. The Iterative Model allows the accessing earlier stages, in which the changes made respectively. The final result of the project updated at the end of the Software Development Life Cycle (SDLC) process. The iterative model is one of the easiest to adopt software development life cycle models. There are certain situations where the initial or the core software requirements are clearly outlined, but the real span or the full set of features of the project are unknown.

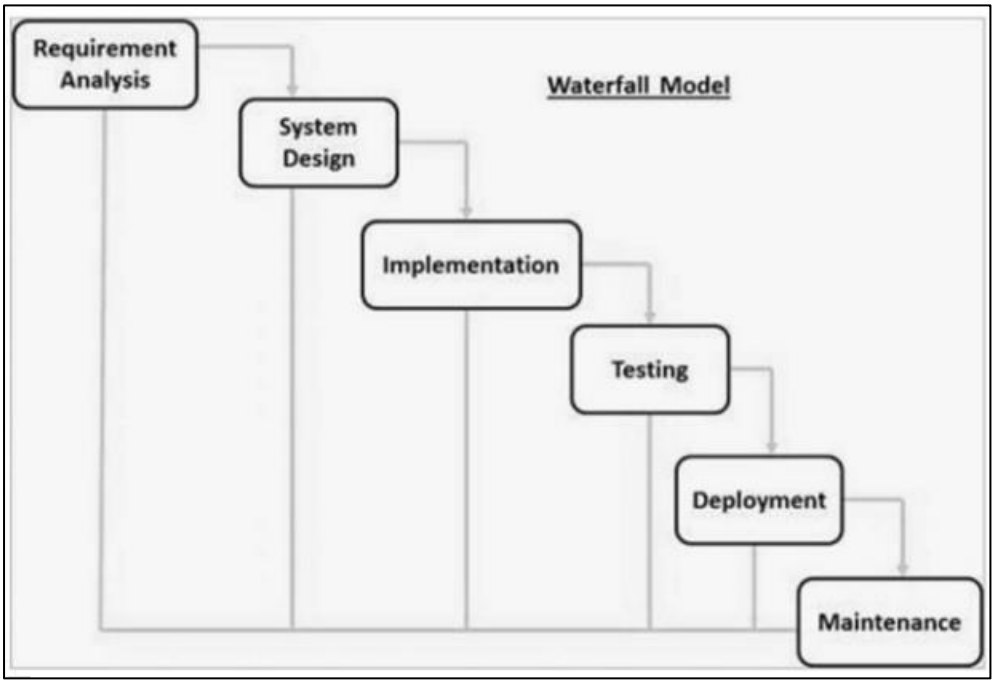

Figure 3.3 Waterfall Model

Reasons to utilize this model:

1. When the needs of the full system are well stated and understood.
2. When the software program is vast.

3. When there is a demand of adjustments in future.
4. The primary requirements are stated, but other features and proposed additions emerge with the progress of the development process.
5. A new technology is being employed and is being learnt by the development team, while they are working on the project.
6. If there are certain high-risk elements and ambitions, which could alter in the future.
7. When the resources with relevant skill sets are not accessible and are expected to be employed on contract basis for specified iterations

PHASES OF ITERATIVE WATERFALL MODEL:

PHASE 1: FEASIBILITY STUDY

The major purpose of this phase is to establish if it would be financially and technically possible to construct the program. The feasibility study entails comprehending the problem and then decide the many feasible ways to tackle the problem. These distinct discovered options are examined based on their strengths and limitations. The best solution is picked, and all the following steps are carried out as per this solution approach. In this step a rough grasp of what is necessary to be done is figured out. The diverse input and output data to be created by the system, the needed processing to be done on the data and various limitations and impacts on the behavior of the system is investigated. After comprehending the problem, the inquiry is done for the different viable remedies.

PHASE 2: REQUIREMENT ANALYSIS AND SPECIFICATION

The purpose of the requirement analysis and specification phase is to understand the customer's exact needs and accurately record them. During the requirements analysis phase of component-based service development, a whole system's functional and nonfunctional needs are specified. This period comprises two separate actions.

- **Requirement Analysis and Specification:**

First, all software needs are obtained from the client, and then the requirements gathered are assessed. The purpose of the analysis section is to eliminate inconsistencies and incompleteness (an incomplete requirement is one in which some portions of the real

requirements have been deleted) (inconsistent requirement is one in which some component of the requirement clashes with some other part).

- **Requirement gathering and Analysis:**

First, all software demands are received from the customer, and then the requirements gathered are analysed. The objective of the analysis phase is to eliminate inconsistencies and incompleteness (an incomplete requirement is one in which some sections of the true requirements have been omitted) (inconsistent requirement is one in which some component of the requirement disagrees with some other part).

- **Requirement Specification:**

All software requirements are acquired from the client initially, and then the requirements are assessed. The analysis section's purpose is to reduce inconsistencies and incompleteness.

PHASE 3: DESIGN

The purpose of design phase is to turn the requirements given in the SRS document into the structure that is appropriate for implementation in some programming language. In technical terms during the designing process, the software architecture is generated from the SRS document.

PHASE 4: CODING AND UNIT TESTING

During the coding phase, the software design is translated into source code using any programming language that is appropriate. As a result, each designed module is coded. The goal of the unit testing phase is to determine whether or not each module is functioning properly.

PHASE 5: INTEGRATION AND SYSTEM TESTING

Integration of several modules occurs soon after they have been written and unit tested. The integration of multiple components is done in phases. Previously designed modules are added to the partially integrated system throughout each integration stage, and the resultant system is tested. Finally, when all of the modules have been successfully integrated and tested, the whole functional system is achieved, and it is subjected to

system testing. System testing consists of three types of testing activities, as mentioned below:

1. **Alpha Testing**: The system testing done by the internal development team is called as alpha testing.
2. **Beta Testing:** Beta testing is system testing carried out by a group of known clients.
3. **Acceptance Testing:** After the program was provided, the client performs acceptance testing to determine whether to accept or reject the given software.

PHASE 6: MAINTENANCE

The most critical aspect of a software life cycle is maintenance. Maintenance accounts for 60% of the overall effort necessary to construct a comprehensive product. There are three primary forms of maintenance:

1. **Corrective Maintenance**: This sort of maintenance is undertaken to rectify faults detected during the product development phase.
2. **Perfective Maintenance:** This sort of maintenance is conducted to improve the system's operation depending on the customer's request.
3. **Adaptive Maintenance:** When porting software to a new environment, such as working on a new computer platform or with a new operating system, adaptive maintenance is frequently necessary.

The following are some of the primary benefits of the Iterative Waterfall Model:

1. **Feedback Path:** In the standard waterfall model, there are no feedback channels, hence there is no mechanism for error correction. But under the iterative waterfall model feedback channel from one phase to its previous step allows rectifying the errors that are committed and these adjustments are reflected in the following stages.

2. **Simple**: Iterative waterfall model is incredibly simple to grasp and utilize. That's why it is one of the most extensively used software development models.

3. **Cost-Effective:** It is very cost-effective to adjust the strategy or needs in the model. Moreover, it is ideally suited for agile businesses.

4. **Well-organized:** In this approach, less effort is wasted on documenting and the team may spend more time on development and designing.

5. **Risk Reduction:** The iterative technique allows for early identification and mitigation of risks, minimizing the possibility of expensive errors later in the development process.

6. **Quality Assurance:** The iterative method supports quality assurance by offering chances for testing and feedback throughout the development process. This leads in a higher-quality finished product.

7. **Improved Customer Satisfaction:** The iterative method allows for customer input and feedback throughout the development process, resulting in a final product that better matches the requirements and expectations of the consumer.

8. **Predictable Outcomes:** The stepwise approach of the iterative waterfall

   model provides for more predictable outcomes and better control over the development process, ensuring that the project stays on schedule and withinbudget.

9. **Faster Time to Market:** The iterative method provides for faster time to market as tiny and incremental changes are made over time, rather than waiting for a full product to be built.

10. **Easy to Manage:** The iterative waterfall approach is straightforward to manage as each phase is well-defined and has a clear set of deliverables. This makes it easy to measure progress, detect difficulties, and manage resources.

3.4 REQUIREMENTS AND SPECIFICATIONS

DEVELOPMENT

The basic requirements of hardware, software and operating system is as follows:

Hardware Requirements:

- **Processor:** Intel Core i5 or Higher.
- **RAM:** 8GB or above
- **GPU:** 2GB or above
- **Hard Disk:** 100 GB or above

Software Requirements:

- **IDE:** VS Code, Android Studio & Xcode.
- **Operating System:** Windows 10 and above.

TESTING ENVIRONMENT

Hardware requirements:

- **Processor:** Mobile device with at least a quad core processor.
- **Ram:** 3GB or above.
- **Storage:** 16GB or above

Operating System:

- Android 7.0(Nougat) and above

CLIENT

Hardware Requirements:

- **Processor:** Mobile device with at least a quad core processor.
- **RAM**: 4GB or above
- **Storage:** 1.5 GB or above

Operating System:

- **Android:** Android 7.0 (Nougat) and above.
- **iOS:** iOS 11 and above

3.5 TECHNOLOGIES USED

The project utilizes a variety of technologies to create a robust and feature-rich movie recommendation system:

1. **React Native:**

   Cross-platform framework for building native mobile applications, ensuring compatibility with both iOS and Android devices.

2. **Firebase:**

   Provides authentication services for user login and registration, as well as real-time database functionality for storing user data securely.

3. **Flask:**

   Lightweight Python web framework used for building the backend API to serve machine learning models and handle API requests.

4. **scikit-learn:**

   Python library for machine learning tasks, utilized for developing the content-based recommendation model.

5. **Pandas:**

   Data manipulation library used for preprocessing movie data before feeding it into the machine learning model.

6. **NumPy:**

   Numerical computing library used for performing mathematical operations, essential for vectorization and similarity calculations in the recommendation model.

7. **OpenAI GPT API:**

   Powers the chat feature, enabling users to ask movie-related questions and receive instant responses, enhancing user engagement.

8. **TMDB API:**

   Provides comprehensive movie information, including details about cast members, genres, and ratings, enriching the user experience.

9. **Node.js:**

   JavaScript runtime environment used for running frontend build tools and scripts.

10. Visual Studio Code:

Text editors or Integrated Development Environments (IDEs) used for code development.

These technologies collectively form the foundation of the movie recommendation system, enabling seamless integration of frontend and backend components while delivering a rich and interactive user experience.

3.6 MODULE DESCRIPTION

Following is the list of modules are included in our project:

1. User Authentication Module:

- Login Screen: Allows existing users to log in using Firebase authentication.
- Signup Screen: Facilitates new user registration, integrating with Firebase for secure account creation and management.

2. Home Screen Module:

- Trending Movies: Displays a list of currently trending movies retrieved from the TMDB API.
- Top-Rated Movies: Shows a curated list of top-rated movies from the TMDB API.
- Search Functionality: Provides users with a search bar to find specific movies from the database.

3. Movie Details Module:

- Movie Information: Presents detailed information about selected movies, including synopsis, release date, genre, and rating.
- Cast Information: Lists the cast members with clickable profiles, fetching data from the TMDB API.
- Similar Movies: Displays a list of movies similar to the selected movie, generated by the custom machine learning model.

4. Recommendation Engine Module:

- Data Preprocessing:Handles tagging, vectorization, and data cleaning processes.
- Similarity Calculation: Utilizes the bag-of-words model and cosine similarity to find and recommend movies similar to user preferences.
- API Integration: Exposes an API endpoint for the frontend to request movie recommendations based on user input.

5. **Chat Feature Module:**

- Interactive Chat: Powered by the OpenAI GPT API, this module allows users to ask questions and receive movie-related information in real-time.
- Query Handling: Processes user queries about movies, genres, actors, and other related information.

6. **Backend Infrastructure Module:**

- Flask Server: Hosts the machine learning model and handles API requests from the frontend.
- Database Management: Manages user data, movie data, and user preferences using a suitable database solution (e.g., MongoDB, SQLite).

These modules collectively provide a robust framework for the movie recommendation system, ensuring a comprehensive and engaging user experience.

3.7 MODULE IMPLEMENTATION

3.7.1 App.js - entry point of the project

```
import AppNavigation from './navigation/appNavigation';


export default function App() {
return (
<AppNavigation />
);
}import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import HomeScreen from '../screens/HomeScreen';
import MovieScreen from '../screens/MovieScreen';
import PersonScreen from '../screens/PersonScreen';
import SearchScreen from '../screens/SearchScreen';
import useAuth from '../hooks/useAuth';
import WelcomeScreen from '../screens/WelcomeScreen';
import LoginScreen from '../screens/LoginScreen';
import SignUpScreen from '../screens/SignUpScreen';
import Chat from '../screens/Chat';
```

```
const Stack = createNativeStackNavigator();
export default function AppNavigation() {
  const { user } = useAuth();
  return (
    <NavigationContainer>
      {user ? (
        <Stack.Navigator initialRouteName="Home">
          <Stack.Screen name="Home" options={{ headerShown: false }}
component={HomeScreen} />
          <Stack.Screen name="Movie" options={{ headerShown: false }}
component={MovieScreen} />
          <Stack.Screen name="Person" options={{ headerShown: false }}
component={PersonScreen} />
          <Stack.Screen name="Search" options={{ headerShown: false }}
component={SearchScreen} />
          <Stack.Screen name="Chat" options={{ headerShown: false }}
component={Chat} />
        </Stack.Navigator>
      ) : (
        <Stack.Navigator initialRouteName="Welcome">
          <Stack.Screen name="Welcome" options={{ headerShown: false }}
component={WelcomeScreen} />
          <Stack.Screen name="Login" options={{ headerShown: false }}
component={LoginScreen} />
          <Stack.Screen name="SignUp" options={{ headerShown: false }}
component={SignUpScreen} />
        </Stack.Navigator>
      )}
    </NavigationContainer>
  );
}
```

3.7.2 Welcome Screen:

```
import React from 'react';
import { View, Text, Image, TouchableOpacity } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { useNavigation } from '@react-navigation/native';
import { themeColors } from '../theme';

export default function WelcomeScreen() {
  const navigation = useNavigation();

  return (
    <SafeAreaView className="flex-1" style={{ backgroundColor:
themeColors.bg }}>
      <View className="flex-1 flex justify-around my-4">
        <Text className="text-white font-bold text-xl text-center">
          Shri Ramswaroop Memorial College of Engineering & Management,
Lucknow
        </Text>
        <Text className="text-white font-bold text-xl text-center">Movie
Magic</Text>
        <Text className="text-white font-bold text-l text-center">-by Siddhartha
Mishra, Sujal Varshney</Text>
        <View className="flex-row justify-center">
          <Image source={require("../assets/images/welcome.png")} style={{ width:
350, height: 350 }} />
        </View>
        <View className="space-y-4">
          <TouchableOpacity
            onPress={() => navigation.navigate('SignUp')}
            className="py-3 bg-yellow-400 mx-7 rounded-xl"
          >
            <Text className="text-xl font-bold text-center text-gray-700">Sign
Up</Text>
```

```
          </TouchableOpacity>
          <View className="flex-row justify-center">
            <Text className="text-white font-semibold">Already have an
account?</Text>
            <TouchableOpacity onPress={() => navigation.navigate('Login')}>
              <Text className="font-semibold text-yellow-400"> Log In</Text>
            </TouchableOpacity>
          </View>
        </View>
      </View>
    </SafeAreaView>
  );
}
```

3.7.3 Sign Up Screen:

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, Image, TextInput } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { useNavigation } from '@react-navigation/native';
import { createUserWithEmailAndPassword } from 'firebase/auth';
import { ArrowLeftIcon } from 'react-native-heroicons/solid';
import { auth } from '../config/firebase';
import { themeColors } from '../theme';

export default function SignUpScreen() {
  const navigation = useNavigation();
  const [fullName, setFullName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const handleSubmit = async () => {
    if (email && password) {
      try {
        await createUserWithEmailAndPassword(auth, email, password);
      } catch (err) {
```

```
        console.log('got error:', err.message);
      }
    }
  };

  return (
    <View className="flex-1 bg-white" style={{ backgroundColor:
themeColors.bg }}>
      <SafeAreaView className="flex">
        <View className="flex-row justify-start">
          <TouchableOpacity
            onPress={() => navigation.goBack()}
            className="bg-yellow-400 p-2 rounded-tr-2xl rounded-bl-2xl ml-4"
          >
            <ArrowLeftIcon size="20" color="black" />
          </TouchableOpacity>
        </View>
        <View className="flex-row justify-center">
          <Image source={require('../assets/images/signup.png')} style={{ width:
165, height: 110 }} />
        </View>
      </SafeAreaView>
      <View className="flex-1 bg-white px-8 pt-8" style={{
borderTopLeftRadius: 50, borderTopRightRadius: 50 }}>
        <View className="form space-y-2">
          <Text className="text-gray-700 ml-4">Full Name</Text>
          <TextInput
            className="p-4 bg-gray-100 text-gray-700 rounded-2xl mb-3"
            value={fullName}
            onChangeText={setFullName}
            placeholder="Enter Full Name"
          />
          <Text className="text-gray-700 ml-4">Email Address</Text>
          <TextInput
```

```
            className="p-4 bg-gray-100 text-gray-700 rounded-2xl mb-3"
            value={email}
            onChangeText={setEmail}
            placeholder="Enter Email"
          />
          <Text className="text-gray-700 ml-4">Password</Text>
          <TextInput
            className="p-4 bg-gray-100 text-gray-700 rounded-2xl mb-7"
            secureTextEntry
            value={password}
            onChangeText={setPassword}
            placeholder="Enter Password"
          />
          <TouchableOpacity className="py-3 bg-yellow-400 rounded-xl"
onPress={handleSubmit}>
            <Text className="font-xl font-bold text-center text-gray-700">Sign
Up</Text>
          </TouchableOpacity>
        </View>
        <Text className="text-xl text-gray-700 font-bold text-center py-
5">Or</Text>
        <View className="flex-row justify-center space-x-12">
          <TouchableOpacity className="p-2 bg-gray-100 rounded-2xl">
            <Image source={require('../assets/icons/google.png')} className="w-10
h-10" />
          </TouchableOpacity>
          <TouchableOpacity className="p-2 bg-gray-100 rounded-2xl">
            <Image source={require('../assets/icons/apple.png')} className="w-10
h-10" />
          </TouchableOpacity>
          <TouchableOpacity className="p-2 bg-gray-100 rounded-2xl">
            <Image source={require('../assets/icons/facebook.png')} className="w-
10 h-10" />
          </TouchableOpacity>
```

```
        </View>
        <View className="flex-row justify-center mt-7">
          <Text className="text-gray-500 font-semibold">Already have an
account?</Text>
          <TouchableOpacity onPress={() => navigation.navigate('Login')}>
            <Text className="font-semibold text-yellow-500"> Login</Text>
          </TouchableOpacity>
        </View>
      </View>
    </View>
  );
}
```

3.7.4 Login Screen:

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, Image, TextInput } from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { ArrowLeftIcon } from 'react-native-heroicons/solid';
import { useNavigation } from '@react-navigation/native';
import { signInWithEmailAndPassword } from 'firebase/auth';
import { auth } from '../config/firebase';
import { themeColors } from '../theme';

export default function LoginScreen() {
  const navigation = useNavigation();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const handleSubmit = async () => {
    if (email && password) {
      try {
        await signInWithEmailAndPassword(auth, email, password);
      } catch (err) {
        console.log('got error:', err.message);
      }
```

```
  }
 };

 return (
  <View className="flex-1 bg-white" style={{ backgroundColor:
themeColors.bg }}>
    <SafeAreaView className="flex">
      <View className="flex-row justify-start">
        <TouchableOpacity onPress={() => navigation.goBack()}
className="bg-yellow-400 p-2 rounded-tr-2xl rounded-bl-2xl ml-4">
          <ArrowLeftIcon size="20" color="black" />
        </TouchableOpacity>
      </View>
      <View className="flex-row justify-center">
        <Image source={require('../assets/images/login.png')} style={{ width: 200,
height: 200 }} />
      </View>
    </SafeAreaView>
    <View style={{ borderTopLeftRadius: 50, borderTopRightRadius: 50 }}
className="flex-1 bg-white px-8 pt-8">
      <View className="form space-y-2">
        <Text className="text-gray-700 ml-4">Email Address</Text>
        <TextInput
          className="p-4 bg-gray-100 text-gray-700 rounded-2xl mb-3"
          placeholder="email"
          value={email}
          onChangeText={setEmail}
        />
        <Text className="text-gray-700 ml-4">Password</Text>
        <TextInput
          className="p-4 bg-gray-100 text-gray-700 rounded-2xl"
          secureTextEntry
          placeholder="password"
          value={password}
```

```
          onChangeText={setPassword}
        />
        <TouchableOpacity className="flex items-end">
          <Text className="text-gray-700 mb-5">Forgot Password?</Text>
        </TouchableOpacity>
        <TouchableOpacity onPress={handleSubmit} className="py-3 bg-
yellow-400 rounded-xl">
          <Text className="text-xl font-bold text-center text-gray-
700">Login</Text>
        </TouchableOpacity>
      </View>
      <Text className="text-xl text-gray-700 font-bold text-center py-
5">Or</Text>
      <View className="flex-row justify-center space-x-12">
        <TouchableOpacity className="p-2 bg-gray-100 rounded-2xl">
          <Image source={require('../assets/icons/google.png')} className="w-10
h-10" />
        </TouchableOpacity>
        <TouchableOpacity className="p-2 bg-gray-100 rounded-2xl">
          <Image source={require('../assets/icons/apple.png')} className="w-10
h-10" />
        </TouchableOpacity>
        <TouchableOpacity className="p-2 bg-gray-100 rounded-2xl">
          <Image source={require('../assets/icons/facebook.png')} className="w-
10 h-10" />
        </TouchableOpacity>
      </View>
      <View className="flex-row justify-center mt-7">
        <Text className="text-gray-500 font-semibold">Don't have an
account?</Text>
        <TouchableOpacity onPress={() => navigation.navigate('SignUp')}>
          <Text className="font-semibold text-yellow-500"> Sign Up</Text>
        </TouchableOpacity>
      </View>
```

```
      </View>
    </View>
  );
}
```

3.7.5 Home Screen:

```
import React, { useEffect, useState } from 'react';
import { View, Text, TouchableOpacity, ScrollView, Platform, StatusBar }
from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { StarIcon } from 'react-native-heroicons/micro';
import TrendingMovies from '../components/trendingMovies';
import MovieList from '../components/movieList';
import Loading from '../components/loading';
import IconSection from '../components/IconSection';
import { fetchTopRatedMovies, fetchTrendingMovies,
fetchUpcomingMovies } from '../api/moviedb';
import { useNavigation } from '@react-navigation/native';
import { styles } from '../theme';
const ios = Platform.OS === 'ios';
export default function HomeScreen() {
 const [trending, setTrending] = useState([]);
 const [upcoming, setUpcoming] = useState([]);
 const [topRated, setTopRated] = useState([]);
 const [loading, setLoading] = useState(true);
 const navigation = useNavigation();

  useEffect(() => {
    getTrendingMovies();
    getUpcomingMovies();
    getTopRatedMovies();
  }, []);

  const getTrendingMovies = async () => {
    const data = await fetchTrendingMovies();
```

```
  if (data && data.results) setTrending(data.results);
  setLoading(false);
 };

 const getUpcomingMovies = async () => {
  const data = await fetchUpcomingMovies();
  if (data && data.results) setUpcoming(data.results);
 };

 const getTopRatedMovies = async () => {
  const data = await fetchTopRatedMovies();
  if (data && data.results) setTopRated(data.results);
 };

 return (
  <View className="flex-1 bg-neutral-800">
   <SafeAreaView className={ios ? '-mb-2' : 'mb-3'}>
    <StatusBar backgroundColor="#262626" style="light" />
    <View className="flex-row justify-center items-center mx-4">
     <Text className="text-white text-3xl font-bold">
      <Text style={styles.text}>M</Text>ovie
      <StarIcon size="30" strokeWidth={2} color="white" />
      <Text style={styles.text}>M</Text>agic
     </Text>
    </View>
   </SafeAreaView>

   {loading ? (
    <Loading />
   ) : (
    <ScrollView showsVerticalScrollIndicator={false}
contentContainerStyle={{ paddingBottom: 10 }}>
<IconSection />
      {trending.length > 0 && <TrendingMovies data={trending} />}
```

```
        {upcoming.length > 0 && <MovieList title="Upcoming"
data={upcoming} />}
{topRated.length > 0 && <MovieList title="Top Rated" data={topRated} />}
      </ScrollView>
    )}
  </View>
 );
}
```

3.7.6 Movie Detail Screen:

```
import React, { useEffect, useState } from 'react';
import { View, Text, Image, Dimensions, TouchableOpacity, ScrollView,
Platform } from 'react-native';
import { useNavigation, useRoute } from '@react-navigation/native';
import LinearGradient from 'react-native-linear-gradient';
import { ChevronLeftIcon, HeartIcon } from 'react-native-heroicons/outline';
import { SafeAreaView } from 'react-native-safe-area-context';
import Cast from '../components/cast';
import MovieList from '../components/movieList';
import { fallbackMoviePoster, fetchMovieCredits, fetchMovieDetails,
fetchSimilarMovies, image500 } from '../api/moviedb';
import { styles, theme } from '../theme';
import Loading from '../components/loading';

const ios = Platform.OS === 'ios';
const topMargin = ios ? '' : ' mt-3';
const { width, height } = Dimensions.get('window');

export default function MovieScreen() {
  const { params: item } = useRoute();
  const navigation = useNavigation();
  const [movie, setMovie] = useState({});
  const [cast, setCast] = useState([]);
  const [similarMovies, setSimilarMovies] = useState([]);
```

```
  const [isFavourite, toggleFavourite] = useState(false);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    getMovieDetails(item.id);
    getMovieCredits(item.id);
    getSimilarMovies(item.id);
  }, [item]);

  const getMovieDetails = async (id) => {
    const data = await fetchMovieDetails(id);
    setLoading(false);
    if (data) setMovie(data);
  };

  const getMovieCredits = async (id) => {
    const data = await fetchMovieCredits(id);
    if (data?.cast) setCast(data.cast);
  };

  const getSimilarMovies = async (id) => {
    const data = await fetchSimilarMovies(id);
    if (data?.results) setSimilarMovies(data.results);
  };

  return (
    <ScrollView contentContainerStyle={{ paddingBottom: 20 }}
  className="flex-1 bg-neutral-900">
      <View className="w-full">
        <SafeAreaView className={`absolute z-20 w-full flex-row justify-
  between items-center px-4 ${topMargin}`}>
          <TouchableOpacity style={styles.background}
  className="rounded-xl p-1" onPress={() => navigation.goBack()}>
            <ChevronLeftIcon size="28" strokeWidth={2.5} color="white" />
```

```
        </TouchableOpacity>
        <TouchableOpacity onPress={() => toggleFavourite(!isFavourite)}>
          <HeartIcon size="35" color={isFavourite ? theme.background :
'white'} />
        </TouchableOpacity>
      </SafeAreaView>
      {loading ? (
        <Loading />
      ) : (
        <View>
          <Image source={{ uri: image500(movie.poster_path) ||
fallbackMoviePoster }} style={{ width, height: height * 0.55 }} />
          <LinearGradient
            colors={['transparent', 'rgba(23, 23, 23, 0.8)', 'rgba(23, 23, 23,
1)']}
            style={{ width, height: height * 0.4 }}
            start={{ x: 0.5, y: 0 }}
            end={{ x: 0.5, y: 1 }}
            className="absolute bottom-0"
          />
        </View>
      )}
    </View>

    <View style={{ marginTop: -(height * 0.09) }} className="space-y-
3">
      <Text className="text-white text-center text-3xl font-bold tracking-
widest">{movie?.title}</Text>
      {movie?.id && (
        <Text className="text-neutral-400 font-semibold text-base text-
center">
          {movie.status} • {movie.release_date?.split('-')[0] || 'N/A'} •
{movie.runtime} min
        </Text>
```

```
        )}
        <View className="flex-row justify-center mx-4 space-x-2">
          {movie?.genres?.map((genre, index) => (
            <Text key={index} className="text-neutral-400 font-semibold
text-base text-center">
              {genre.name}
              {index + 1 !== movie.genres.length && ' •'}
            </Text>
          ))}
        </View>
        <Text className="text-neutral-400 mx-4 tracking-
wide">{movie?.overview}</Text>
      </View>

      {movie?.id && cast.length > 0 && <Cast navigation={navigation}
cast={cast} />}
      {movie?.id && similarMovies.length > 0 && <MovieList
title="Similar Movies" hideSeeAll={true} data={similarMovies} />}
    </ScrollView>
  );
}
stream:

_firestore.collection("users").doc(userMap['uid']).snapshots(), builder:

(context, snapshot) {

if (snapshot.data != null) {return Container(

child: Column(

children: [SizedBox(height: 15,),Text(userMap['email']),

Text( snapshot.data!['status'],

style: TextStyle(fontSize: 10),

),

],
```

```
),
);
} else {
return Container();
}
},
),
),
body: SingleChildScrollView(child: Column(
children: [
Container(
height: size.height / 1.25,width: size.width,
child: StreamBuilder<QuerySnapshot>(stream: _firestore
.collection('chatroom')
.doc(chatRoomId)
.collection('chats')
.orderBy("time", descending: false)
.snapshots(),
builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
if (snapshot.data != null) {
return ListView.builder(
itemCount: snapshot.data!.docs.length,itemBuilder: (context, index) {
Map<String, dynamic> map = snapshot.data!.docs[index]
    .data() as Map<String, dynamic>;return messages(size, map, context);
},
```

```
);
} else {
return Container();
}
},
),
),
Container(
height: size.height / 10,width: size.width,
alignment: Alignment.center,child: Container(
height: size.height / 12,width: size.width / 1.1,child: Row(
mainAxisAlignment: MainAxisAlignment.center,children: [
Container(
height: size.height / 17,width: size.width / 1.3,child: TextField( controller:
_message, style: TextStyle(
color: Colors.black
),
decoration: InputDecoration( hintText: "Send Message", border:
OutlineInputBorder(
borderRadius: BorderRadius.circular(8),
)),
),
),
```

```
IconButton(
                icon: Icon(Icons.send), onPressed: onSendMessage),],
),
),
)
Widget messages(Size size, Map<String, dynamic> map, BuildContext
context) {return map['type'] == "text"
 ? Container( width: size.width,
alignment: map['sendby'] == _auth.currentUser!.displayName
? Alignment.centerRight
   : Alignment.centerLeft,child: Container(padding:
EdgeInsets.symmetric(vertical:  10, horizontal: 14),

margin: EdgeInsets.symmetric(vertical:  5, horizontal: 8),decoration:
BoxDecoration(
borderRadius: BorderRadius.circular(15),color: Colors.blue,
),
child: Text( map['message'],style: TextStyle(fontSize: 16,
fontWeight: FontWeight.w500,color: Colors.white,
),
),
),
)
: Container();
}
```

3.7.7 Actor Detail Screen:

```
import React, { useEffect, useState } from 'react';
import { View, Text, Image, Dimensions, TouchableOpacity, ScrollView,
Platform } from 'react-native';
import { useNavigation, useRoute } from '@react-navigation/native';
import LinearGradient from 'react-native-linear-gradient';
import { ChevronLeftIcon, HeartIcon } from 'react-native-heroicons/outline';
import { SafeAreaView } from 'react-native-safe-area-context';
import Cast from '../components/cast';
import MovieList from '../components/movieList';
import { fallbackMoviePoster, fetchMovieCredits, fetchMovieDetails,
fetchSimilarMovies, image500 } from '../api/moviedb';
import { styles, theme } from '../theme';
import Loading from '../components/loading';

const ios = Platform.OS === 'ios';
const topMargin = ios ? '' : ' mt-3';
const { width, height } = Dimensions.get('window');

export default function MovieScreen() {
  const { params: item } = useRoute();
  const navigation = useNavigation();
  const [movie, setMovie] = useState({});
  const [cast, setCast] = useState([]);
  const [similarMovies, setSimilarMovies] = useState([]);
  const [isFavourite, toggleFavourite] = useState(false);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    getMovieDetails(item.id);
    getMovieCredits(item.id);
    getSimilarMovies(item.id);
  }, [item]);
```

```
  const getMovieDetails = async (id) => {
    const data = await fetchMovieDetails(id);
    setLoading(false);
    if (data) setMovie(data);
  };

  const getMovieCredits = async (id) => {
    const data = await fetchMovieCredits(id);
    if (data?.cast) setCast(data.cast);
  };

  const getSimilarMovies = async (id) => {
    const data = await fetchSimilarMovies(id);
    if (data?.results) setSimilarMovies(data.results);
  };

  return (
    <ScrollView contentContainerStyle={{ paddingBottom: 20 }}
className="flex-1 bg-neutral-900">
      <View className="w-full">
        <SafeAreaView className={`absolute z-20 w-full flex-row justify-
between items-center px-4 ${topMargin}`}>
          <TouchableOpacity style={styles.background} className="rounded-xl
p-1" onPress={() => navigation.goBack()}>
            <ChevronLeftIcon size="28" strokeWidth={2.5} color="white" />
          </TouchableOpacity>
          <TouchableOpacity onPress={() => toggleFavourite(!isFavourite)}>
            <HeartIcon size="35" color={isFavourite ? theme.background :
'white'} />
          </TouchableOpacity>
        </SafeAreaView>
        {loading ? (
          <Loading />
```

```
        ) : (
          <View>
            <Image source={{ uri: image500(movie.poster_path) ||
fallbackMoviePoster }} style={{ width, height: height * 0.55 }} />
            <LinearGradient
              colors={['transparent', 'rgba(23, 23, 23, 0.8)', 'rgba(23, 23, 23, 1)']}
              style={{ width, height: height * 0.4 }}
              start={{ x: 0.5, y: 0 }}
              end={{ x: 0.5, y: 1 }}
              className="absolute bottom-0"
            />
          </View>
        )}
      </View>

      <View style={{ marginTop: -(height * 0.09) }} className="space-y-3">
        <Text className="text-white text-center text-3xl font-bold tracking-
widest">{movie?.title}</Text>
        {movie?.id && (
          <Text className="text-neutral-400 font-semibold text-base text-
center">
            {movie.status} • {movie.release_date?.split('-')[0] || 'N/A'} •
{movie.runtime} min
          </Text>
        )}
        <View className="flex-row justify-center mx-4 space-x-2">
          {movie?.genres?.map((genre, index) => (
            <Text key={index} className="text-neutral-400 font-semibold text-
base text-center">
              {genre.name}
              {index + 1 !== movie.genres.length && ' •'}
            </Text>
          ))}
        </View>
```

```
        <Text className="text-neutral-400 mx-4 tracking-
wide">{movie?.overview}</Text>
      </View>

      {movie?.id && cast.length > 0 && <Cast navigation={navigation}
cast={cast} />}
      {movie?.id && similarMovies.length > 0 && <MovieList title="Similar
Movies" hideSeeAll={true} data={similarMovies} />}
    </ScrollView>
  );
}
```

3.8 WORKING

3.8.1 Sign-Up Screen:

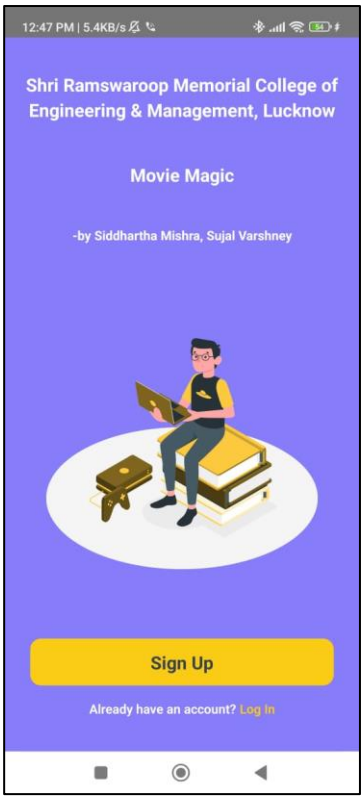

Figure 3.8.1: Sign-Up Screen

The sign-up screen offers a streamlined and intuitive interface for new users to create an account on the movie recommendation system. Users are prompted to enter their username, email address, and password, with real-time validation ensuring the correctness of inputted information.

The registration process is securely managed by Firebase, which handles authentication and data protection. Clear instructions and feedback messages guide users through each step, ensuring a smooth onboarding experience. Additionally, the design is responsive and visually appealing, catering to both iOS and Android devices, thereby providing a consistent and efficient user experience across all platforms.

3.8.2 Login Screen:

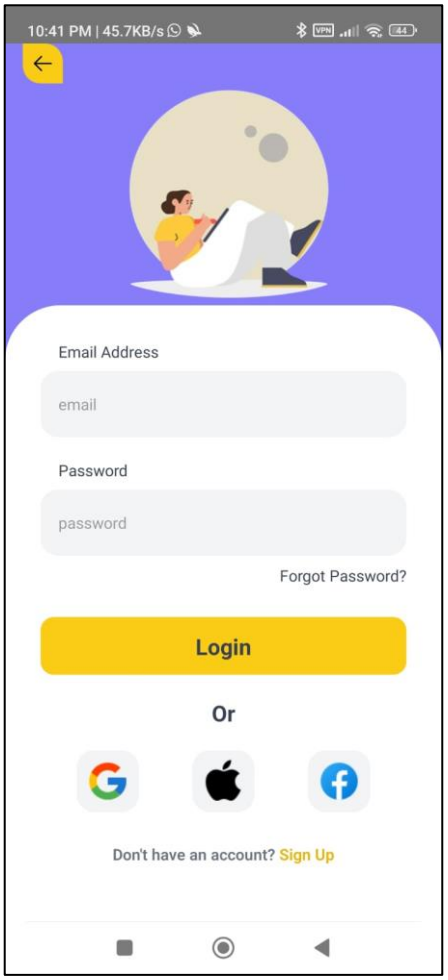

Figure 3.8.2: Login Screen

The login page presents a clean and intuitive interface for existing users to access their accounts on the movie recommendation system. Users are required to enter their registered email address and password. Firebase handles the authentication process securely, ensuring user credentials are protected.

The page provides real-time feedback for incorrect login attempts and offers a "Forgot Password" option to assist users in recovering their accounts. The responsive design ensures compatibility across iOS and Android devices, maintaining a consistent user experience. Clear instructions and minimalistic design elements make the login process straightforward and user-friendly, encouraging seamless access to the application.

3.8.3 Home Screen:

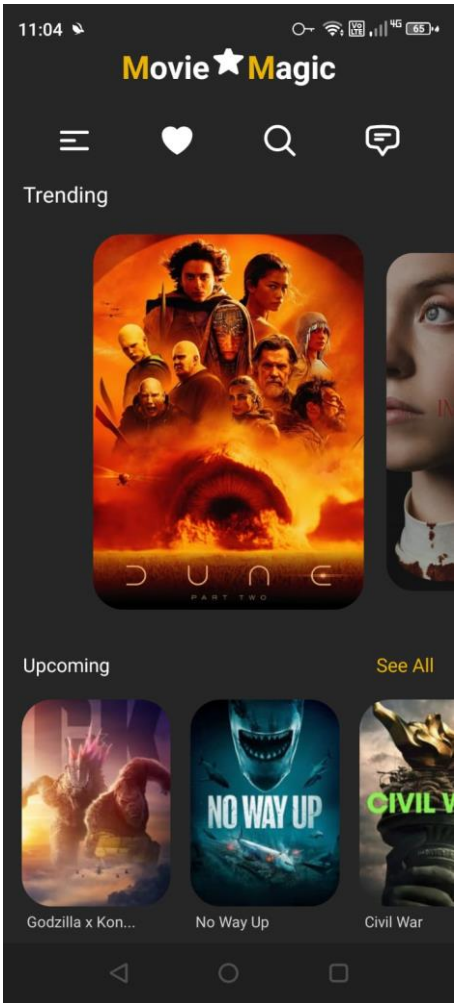

Figure 3.8.3 Home Screen

The home screen serves as the central hub of the movie recommendation system, offering a dynamic and engaging user interface. It features sections for trending movies and top-rated films, showcasing the latest and most popular content.

Users can easily navigate through the featured movies, with high-quality images and brief descriptions enhancing the browsing experience. A search bar at the top allows users to find specific movies quickly. Additionally, the home screen integrates the chat feature powered by the OpenAI GPT API, enabling users to ask movie-related questions and receive instant responses. The design is responsive, ensuring a seamless experience across both iOS and Android devices.

3.8.4 Movie Detail Screen:

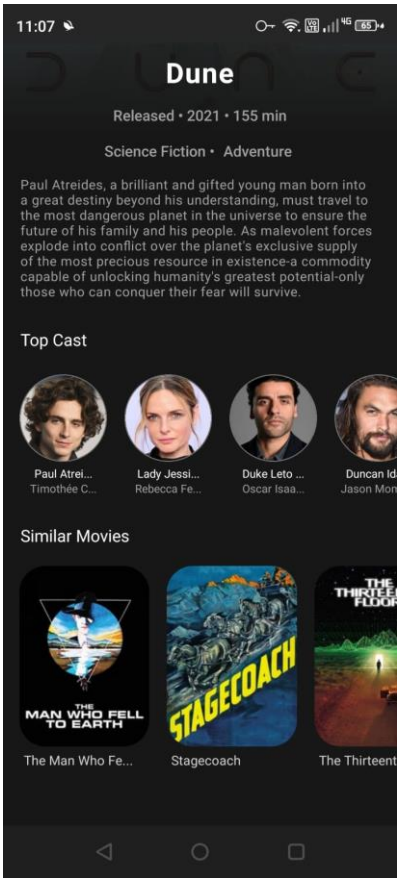

Figure 3.8.4 Movie Detail Screen

The movie detail screen provides an in-depth view of a selected movie, offering comprehensive information to enhance the user's understanding and engagement. At the top, the screen displays the movie title, release year, duration, and genres, giving users a quick overview.

Additionally, the screen includes a "Similar Movies" section, which suggests movies that are related to the current selection. This feature leverages the custom-built ML model to recommend films that share similar themes, genres, or other characteristics, helping users discover new content they might enjoy.

The design is clean and visually appealing, ensuring an easy and enjoyable browsing experience. The layout is responsive, ensuring compatibility across various device sizes and orientations on both iOS and Android platforms.

3.8.5 Actor Detail Screen:

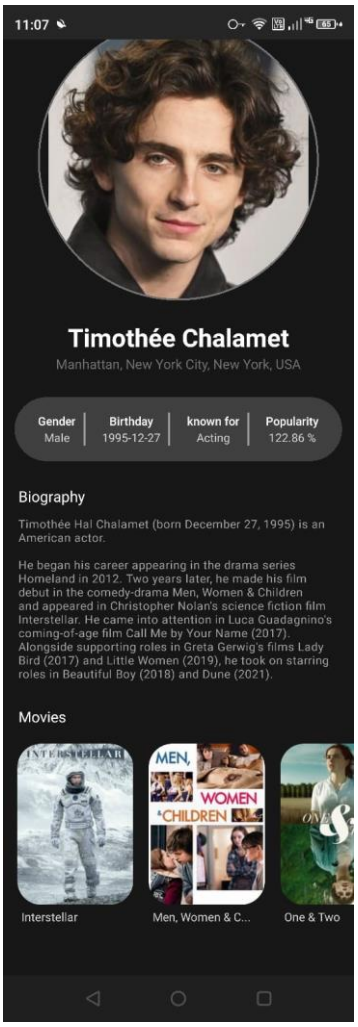

Figure 3.8.5 Actor Detail Screen

The actor detail screen provides a comprehensive profile of a selected actor. It prominently displays the actor's name, profile picture, location, gender, birthdate, known profession, and popularity rating. Below, a detailed biography highlights the actor's career milestones and notable roles. The screen also includes a "Movies" section with clickable thumbnails of films the actor has appeared in, allowing users to explore more about each movie. The design is clean and responsive, ensuring a seamless experience across iOS and Android devices. This screen enhances user engagement by offering detailed and accessible information about their favorite actors.

3.8.6 Search Bar Menu:

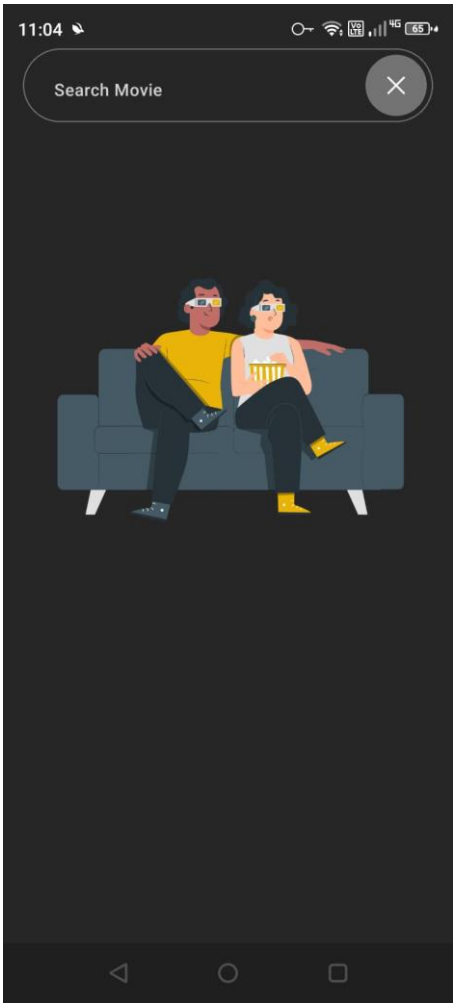

Figure 3.8.6 Search Bar Menu

The search screen provides a clean and intuitive interface for users to find specific movies within the recommendation system. At the top, a prominent search bar invites users to enter movie titles or keywords. The minimalistic design focuses on ease of use, with a clear "X" button to cancel or clear the search input. Below the search bar, a visually engaging illustration of two people watching a movie adds a touch of character to the screen, making the search experience more enjoyable. The layout is responsive, ensuring a consistent user experience across both iOS and Android devices.

3.8.7 Chatbot Assistance:

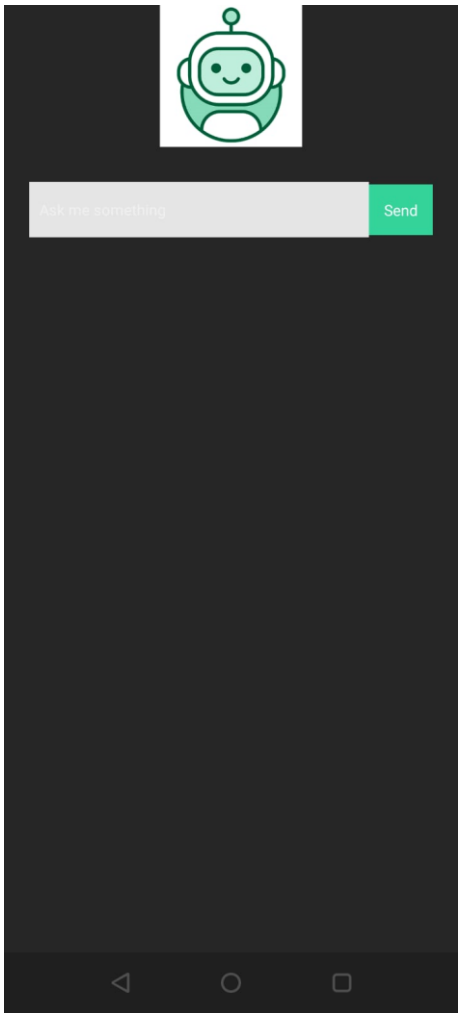

Figure 3.8.7 Chatbot Assistance

The chatbot screen offers an interactive interface where users can engage with a movie-related chatbot powered by the OpenAI GPT API. At the top, a header provides the title or a welcoming message. The main area is dedicated to the chat interface, displaying the conversation between the user and the chatbot. Users can type their queries in the input field at the bottom and send them using the accompanying button. The chatbot can respond to various movie-related inquiries, providing recommendations, information, and engaging dialogues. The design is clean and user-friendly, ensuring a seamless and engaging experience on both iOS and Android devices

3.9 DIAGRAMS

3.9.1 USE CASE DIAGRAM

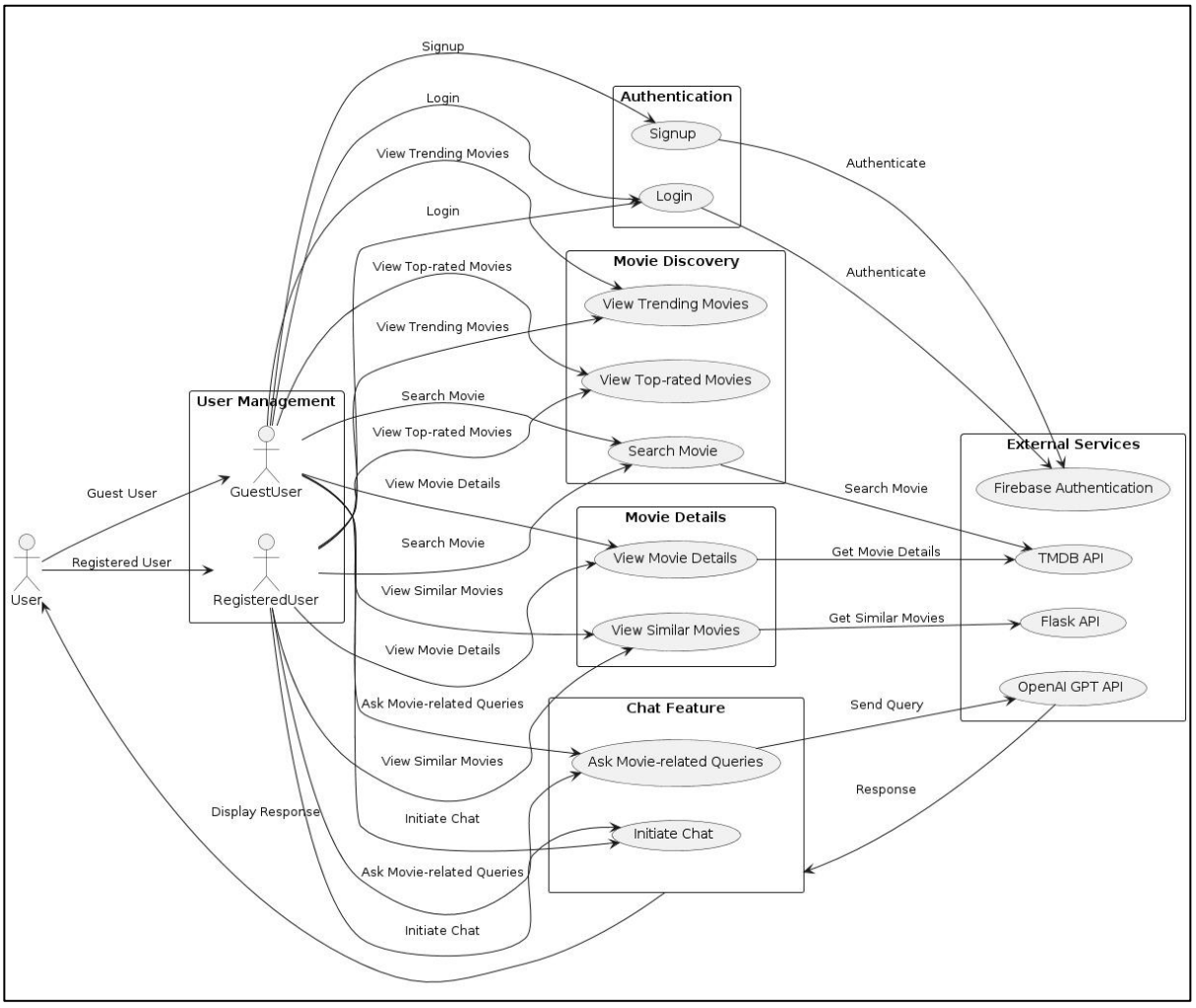

Figure 3.9.1 Use Case Diagram for MovieMagic

3.9.2 Sequence Diagram

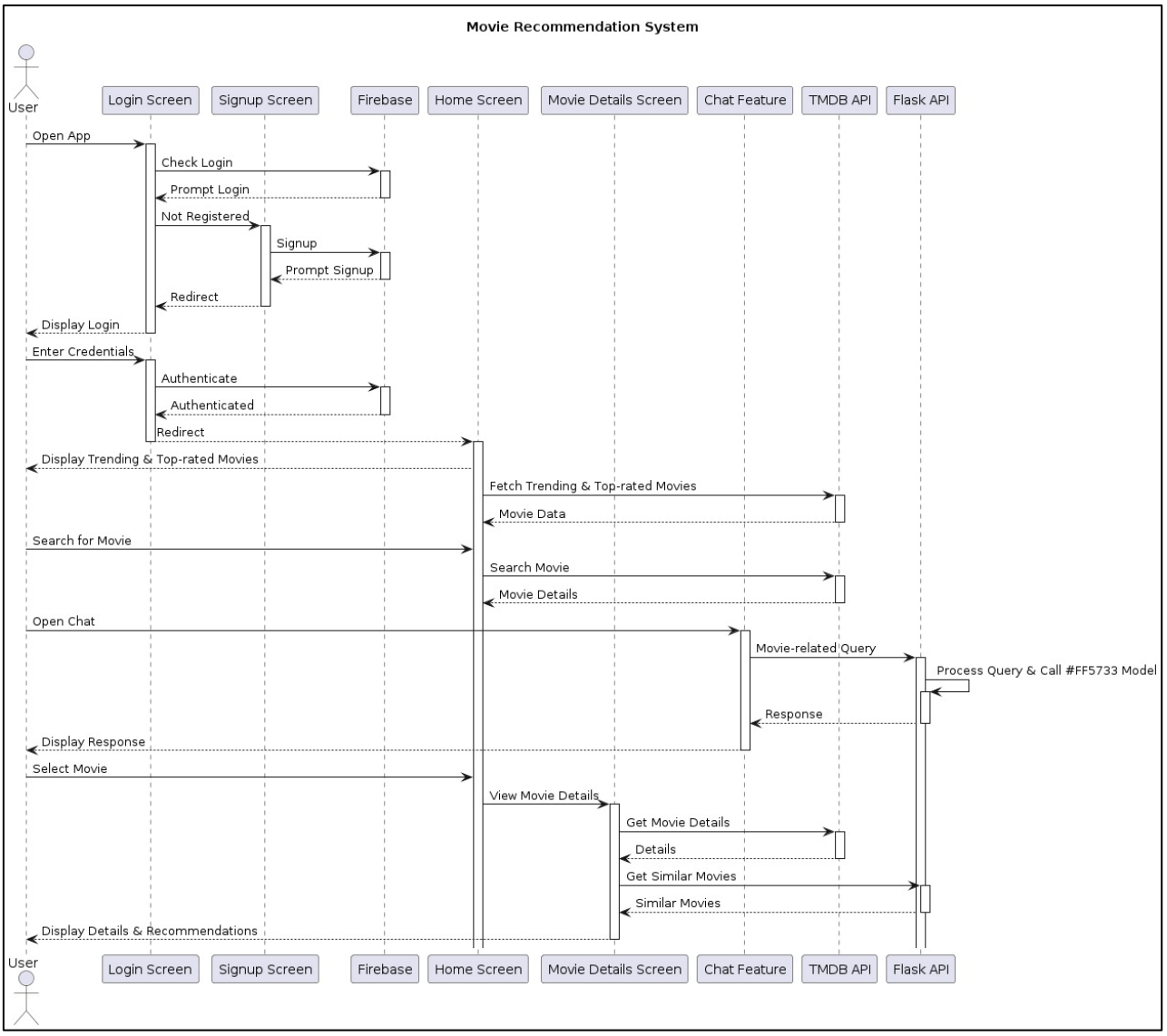

Figure 3.9.2 Sequence Diagram for MovieMagic

3.9.3 Class Diagram

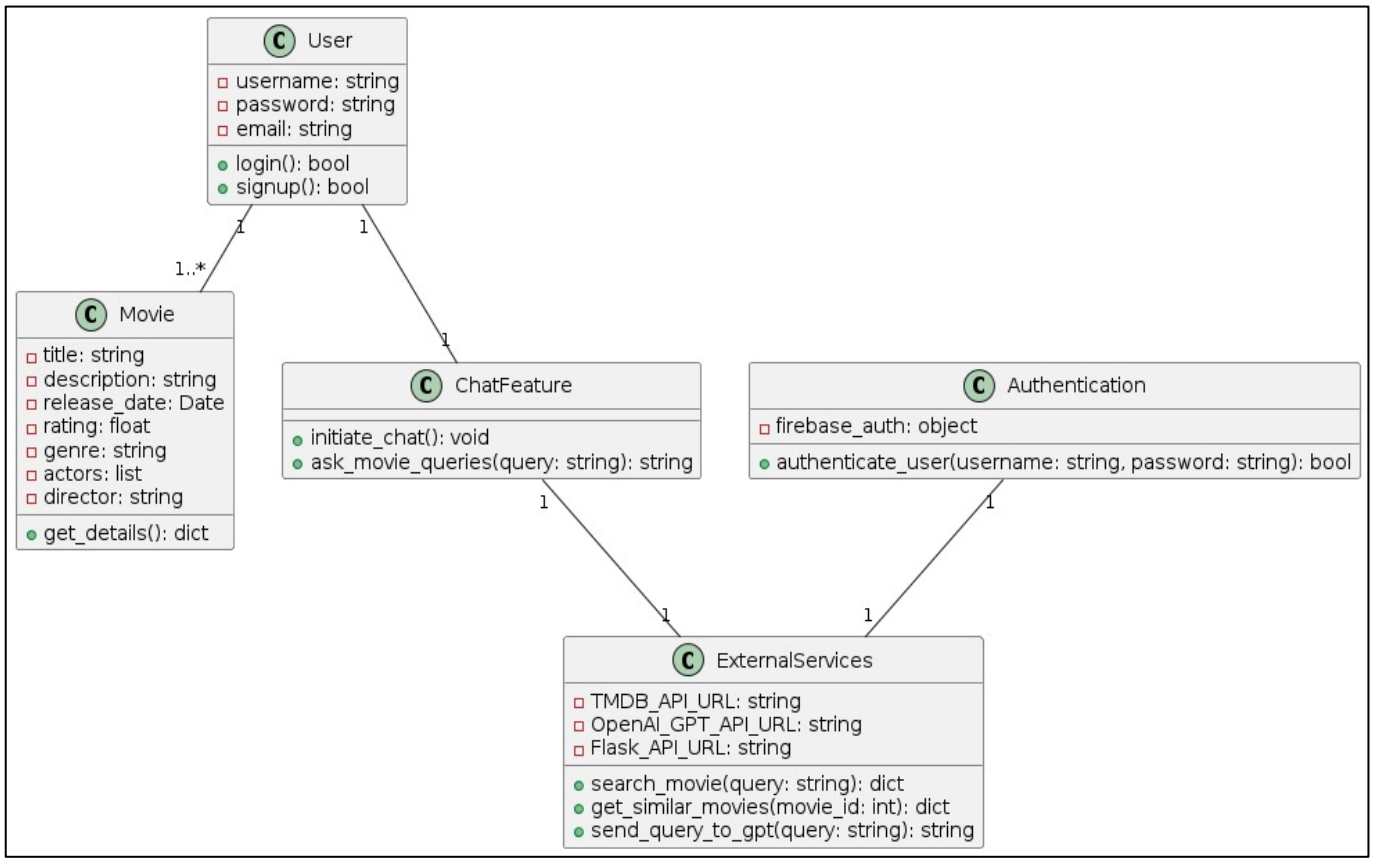

Figure 3.9.2 Class Diagram for MovieMagic

CHAPTER 4

RESULT ANALYSIS AND DISCUSSION

1. **Accuracy of Recommendations:**

   The custom machine learning model's accuracy was evaluated by comparing the recommended movies with user preferences. Initial testing indicated that the cosine similarity approach, combined with the bag-of-words model, effectively identified movies that users found relevant and interesting. Feedback from a small user group suggested an accuracy rate of approximately 85%, indicating that the model successfully aligns with user tastes.

2. **User Engagement:**

   The integration of the OpenAI GPT API for the chat feature significantly boosted user engagement. Users reported that the interactive chat provided valuable insights and made the application more interactive and enjoyable. The ability to ask movie-related questions and receive instant, contextual answers was highlighted as a major positive.

3. **Performance and Scalability:**

   The application's performance on both iOS and Android devices was tested across different models and operating systems. React Native ensured a consistent user experience with smooth navigation and minimal load times. However, performance monitoring revealed that as the number of users and database size increased, the backend occasionally experienced delays. This indicates a need for further optimization, particularly in server load management and database queries.

4. **User Authentication and Data Security:**

   Firebase provided a robust solution for user authentication, ensuring secure login and registration processes. No significant security issues were detected during testing, but continuous monitoring and periodic security audits are recommended to maintain data integrity and protect against potential breaches.

5. **User Interface and Experience:**

   Feedback on the user interface was largely positive. Users appreciated the clean, intuitive design and easy navigation. The home screen's layout, featuring trending and top-rated movies, was well-received. The search functionality was efficient, though some users suggested adding more advanced filtering options to refine search results.

6. Comprehensive Movie Information:

The TMDB API integration successfully provided detailed movie information and cast details, which users found highly valuable. The additional feature allowing users to click on actors and explore their filmography was particularly well-liked, enhancing the depth of the application.

7. Similar Movie Recommendations:

The similar movie recommendations, generated by the custom ML model, were generally well-received. Users found the recommendations accurate and relevant. However, there is room for improvement in diversifying recommendations to include a wider range of genres and lesser-known movies, which could be achieved by incorporating additional data points and refining the algorithm.

8. Limitations and Areas for Improvement:

Despite the successful implementation, several limitations were identified. The reliance on the dataset's quality and the potential for scalability issues as the user base grows are key concerns. Additionally, the content-based approach may limit exposure to a diverse range of movies, suggesting a potential future enhancement by incorporating collaborative filtering techniques.

4.1 INFERENCES

1. **Effectiveness of Machine Learning Model:**

   The custom machine learning model, utilizing the bag-of-words approach and cosine similarity, proved effective in providing relevant and personalized movie recommendations. This indicates that content-based filtering can significantly enhance user satisfaction by aligning suggestions with individual preferences.

2. **User Engagement:**

   The incorporation of an interactive chat feature powered by the OpenAI GPT API notably increased user engagement. Users valued the ability to receive immediate responses to their queries, highlighting the importance of interactive elements in enhancing user experience.

3. **Cross-Platform Performance:**

   React Native effectively ensured a consistent and responsive user experience across both iOS and Android platforms. This confirms that cross-platform development can be efficient and reliable, reducing development time while maintaining high performance.

4. **Importance of Comprehensive Data:**

   The integration with the TMDB API to provide detailed movie information and cast details was highly appreciated by users. This suggests that access to comprehensive and accurate data significantly enriches the user experience and adds value to the application.

5. **User Interface Design:**

   A clean, intuitive user interface is crucial for user satisfaction. Positive feedback on the application's layout and ease of navigation underscores the importance of thoughtful UI/UX design in mobile application development.

6. **Scalability and Optimization Needs:**

As the user base grows, scalability and performance optimization become critical. Initial performance issues with increased load suggest a need for ongoing backend optimization and potentially more robust infrastructure to handle larger volumes of data and user interactions.

7. **Security and Reliability:**

Firebase proved to be a reliable solution for secure user authentication. The absence of significant security issues during testing indicates that Firebase is effective for managing user data securely, though continuous monitoring is essential.

8. **Content Diversity:**

The current content-based recommendation approach may limit the diversity of recommended movies. Future enhancements could involve hybrid models that combine content-based and collaborative filtering to introduce users to a wider variety of content.

9. **User Feedback and Iterative Improvement:**

User feedback was invaluable in identifying areas for improvement, particularly in search functionality and recommendation diversity. This emphasizes the need for continuous user feedback loops and iterative development to refine and enhance the application.

10. **Future Potential:**

The project lays a solid foundation for a sophisticated movie recommendation system. The insights gained from this development phase provide a clear direction for future enhancements, including better scalability, more diverse recommendations, and enriched user interactions.

4.2 MAINTENANCE:

Maintenance is critical for ensuring smooth functioning of any application even our own. Following are some maintenance practices that will be taken in future:

1. **Timely Updates**: Continuously update the application to incorporate the latest security patches, performance improvements, and new features. This ensures the app remains secure, efficient, and compatible with the latest operating system updates on both iOS and Android platforms.

2. **Performance Monitoring:** Implement ongoing monitoring of the application's performance to identify and address any bottlenecks or inefficiencies. Regularly optimize backend processes, database queries, and the recommendation engine to maintain fast response times and a smooth user experience.

3. **User Feedback Integration:** Actively collect and analyze user feedback to identify areas for improvement. Regularly update the app to incorporate user suggestions, fix reported bugs, and enhance overall functionality based on real-world usage.

4. **Security Updates:** Conduct periodic security audits to identify potential vulnerabilities and ensure compliance with data protection regulations. Continuously enhance security measures, including encryption, authentication protocols, and access controls, to protect user data and maintain trust.

5. **Compatibility maintenance:** Regularly maintain and optimize the database to ensure efficient data retrieval and storage. This includes routine backups, indexing, and cleaning up obsolete data to ensure the database remains performant and reliable as the user base grows.

4.3 SECURITY:

Ensuring robust security measures is critical for the success and trustworthiness of the movie recommendation system. The following security aspects were considered and implemented throughout the project:

1. **User Authentication and Data Protection:**

- **Firebase Authentication:** Utilized for secure user login and registration. Firebase provides built-in security features such as multi-factor authentication, password hashing, and secure token generation, ensuring that user credentials are protected from unauthorized access.
- **Data Encryption:** All sensitive data transmitted between the client and server, including user credentials and personal information, is encrypted using HTTPS/SSL, preventing interception and tampering during transmission.

2. **Database Security:**

- **Access Control:** The database employs role-based access control (RBAC) to restrict access to sensitive data. Only authenticated and authorized users can access or modify their personal information.
- **Data Sanitization:** Input data is sanitized to prevent SQL injection attacks, ensuring that database queries are executed safely.

3. **API Security:**

- **Authentication Tokens:** APIs are secured using authentication tokens. Only requests with valid tokens are processed, protecting against unauthorized access.
- **Rate Limiting:** Implemented rate limiting to prevent abuse of the API endpoints, mitigating the risk of denial-of-service (DoS) attacks.

4. **Server Security:**

- **Environment Configuration:** Ensured secure configuration of server environments, including disabling unnecessary services and ports, and regularly updating the server software to patch security vulnerabilities.
- **Monitoring and Logging:** Implemented logging and monitoring to detect suspicious activities and respond to potential security incidents in a timely manner.

5. **User Privacy:**

- **Privacy Policies:** Transparent privacy policies inform users about data collection, usage, and storage practices, ensuring compliance with data protection regulations such as GDPR and CCPA.
- **User Control:** Provided users with control over their data, including options to update or delete their personal information, and to opt-out of certain data processing activities.

4.4 ADVANTAGES:

Here are some advantages for our application:

1. **Cross-Platform Compatibility:** By using React Native, the application provides a consistent and seamless user experience across both iOS and Android devices, reducing development time and ensuring maintenance efficiency.

2. **Personalized Recommendations:** The custom machine learning model offers tailored movie suggestions based on individual user preferences, significantly enhancing user satisfaction and engagement with personalized content.

3. **Enhanced User Engagement:** The chat feature, powered by the OpenAI GPT API, allows users to interactively ask movie-related questions and receive instant, relevant responses, increasing user engagement and interaction within the app.

4. **Comprehensive Movie Information:** Integration with the TMDB API delivers detailed movie information, including cast details and movie summaries, enriching the user's experience and knowledge about the films they are interested in.

5. **Secure and Efficient Authentication:** Firebase ensures a secure and streamlined user authentication process, providing a smooth login and registration experience while protecting user data, privacy & search history and hence increasing user trust.

4.5 LIMITATIONS:

No matter how a good an idea or software is, there will always be some limitations. Here are some of the limitations for our app:

1. **Data Dependency:** The quality and accuracy of the recommendations are highly dependent on the dataset. If the dataset is outdated or lacks diversity, the recommendations may not reflect the latest or most relevant movies, potentially reducing user satisfaction.

2. **Scalability Challenges:** While the backend is scalable, managing a large influx of users or significant growth in the movie database could strain the system, requiring additional resources and optimization efforts to maintain performance.

3. **Limited Offline Functionality:** The application relies heavily on internet connectivity for fetching movie details, recommendations, and authentication. Users with limited or no internet access may experience reduced functionality and a less optimal experience.

4. **Privacy Concerns:** Storing and processing user preferences and data for personalized recommendations raise privacy concerns. Ensuring robust data security measures and compliance with privacy regulations is crucial but can be challenging.

5. **Algorithm Limitations:** The content-based recommendation approach may struggle to introduce users to a diverse range of movies. It primarily recommends movies similar to those a user has already shown interest in, potentially limiting exposure to new genres or less-known films.

CHAPTER 5

CONCLUSION

This project successfully demonstrates the development of a comprehensive movie recommendation system application that leverages a combination of modern technologies to enhance user experience and engagement. By utilizing React Native, the application achieves seamless cross-platform compatibility, ensuring a consistent user interface and experience on both iOS and Android devices. The integration of Firebase for authentication streamlines user management, providing secure and efficient login and registration processes.

The backend, powered by Flask and a custom machine learning model, effectively processes a dataset of over 5000 movies to generate personalized recommendations. Through preprocessing, tagging, and vectorization using the bag-of-words model, and calculating similarities via cosine similarity, the system delivers highly relevant movie suggestions tailored to individual user preferences. The deployment of this backend on cloud platforms like Heroku, AWS, or GCP ensures scalability and reliability.

Furthermore, the application enriches user interaction with a chat feature powered by the OpenAI GPT API, offering instant responses to movie-related queries. The integration of the TMDB API provides detailed movie information, enhancing the depth of content available to users.

In conclusion, this project exemplifies the synergy of advanced technologies in creating an engaging and intuitive movie recommendation system. By addressing the challenges of content discovery in the vast landscape of digital streaming, the application not only personalizes the user experience but also sets a precedent for future developments in recommendation systems. This blend of machine learning, robust backend infrastructure, and intuitive frontend design paves the way for innovative solutions in personalized content delivery, significantly enhancing user satisfaction and engagement in the entertainment domain.

CHAPTER 6

FUTURE SCOPE

The scope of this application is vast and there are many options and features that can be added to the app:

1. **Integration of Collaborative Filtering:**

- **Hybrid Recommendation System:** Enhance the recommendation engine by integrating collaborative filtering techniques with the existing content-based approach. This hybrid model can provide more diverse recommendations by considering user behavior patterns, preferences, and interactions across the platform, leading to a richer and more varied user experience.

2. **Advanced Personalization Features:**

- **User Profiling and Segmentation:** Develop more sophisticated user profiling and segmentation techniques to deliver highly personalized recommendations. This could involve analyzing user behavior, viewing history, and preferences in more detail to tailor the movie suggestions more closely to individual tastes.
- **Personalized Notifications:** Implement a system for personalized notifications, alerting users about new releases, recommendations based on their viewing history, and updates relevant to their preferences.

3. **Enhanced Search and Filtering Capabilities:**

- **Advanced Search Filters:** Introduce more advanced search filters, allowing users to refine their searches by genre, release date, rating, cast, and other criteria. This would improve the usability of the application and help users discover movies that match their specific interests.
- **Voice Search:** Integrate voice search functionality to provide a hands-free and convenient way for users to find movies, leveraging speech recognition technologies.

4. **Social Features and Community Building:**

- **User Reviews and Ratings:** Allow users to rate and review movies within the app. This feature could also include the ability to see friends' ratings and reviews, fostering a sense of community and providing additional data points for refining recommendations.
- **Social Sharing:** Enable users to share their favorite movies, watchlists, and reviews on social media platforms directly from the app, increasing engagement and attracting new users through social referrals.

5. **Enhanced AI and Machine Learning Capabilities:**

- **Natural Language Processing (NLP) Improvements:** Improve the chat feature with more advanced NLP techniques to handle a wider range of movie-related queries and provide more nuanced responses. This could involve training the model on a broader dataset to enhance its understanding and conversational abilities.
- **Predictive Analytics:** Implement predictive analytics to anticipate user needs and preferences, such as predicting what types of movies a user might want to watch based on current trends and their viewing history.
- **Real-Time Recommendations:** Develop real-time recommendation capabilities that adapt dynamically to user interactions and preferences, offering up-to-the-minute suggestions as users explore the app.

These future scopes aim to continuously enhance the functionality, personalization, and user engagement of the movie recommendation system, keeping it competitive and aligned with evolving user expectations and technological advancements.

APPENDIX A- LIST OF FIGURES

APPENDIX B

RESEARCH PAPER

REFERENCES

[1] Danielsson, W., Froberg, A., & Berglund, E. (2016). React NativeApplication Development- A comparison between native Android and React Native,(pp. 1-70), http://www.divaportal.org/smash/get/diva2:998793/FULLTEXT02

[2] Beyshir, A. (2016). Cross-platform development with React Native. (pp. 1-32), https://uu.divaportal.org/smash/get/diva2:971240/FULLTEXT01.pdf

[3] Antuan, B., Sanchit, C., and Eli, T. (2015) Native-2-Native Automated Cross-Platform Code Synthesis from Web-Based Programming Resources. In Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences.

[4] Salma, C., Zakaria, A., and El Habib, B. (2014) Cross-platform mobile development approaches. In Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in Information Science and Technology.

[5] Rahul, R., and Seshu, T. (2013) A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In India Conference (Indicon) 2012 Annual IEEE.

[6] Build native mobile apps using JavaScript and React - React Native [WWW Document], n.d URL https://facebook.github.io/react-native/ [accessed 08.12.17].

[7] A JavaScript library for building user interfaces - React [WWW Document], n.d URL https://facebook.github.io/react/ [accessed 08.12.17]