NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA

KNOWLEDGE * CHARACTER * UNITY

PROJECT REPORT

on

ONLINE MOVIE BOOKING PORTAL USING MERN STACK

Submitted in partial fulfillment of the requirement for the award of Degree of

Bachelor of Engineering

in

Information Science and Engineering

Submitted by:

Apar Mahajan	1NT20IS027
Devank Gupta	1NT20IS047

Under the Guidance of

Mr. Prashanth B S, Mrs. Vani K. S, Mrs. Evangeline R C

Assistant Professors, Dept. of ISE, NMIT

NITTE EDUCATION TRUST

Department of Information Science and Engineering **(Accredited by NBA Tier-1)**

2022-2023

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM)

Department of Information Science and Engineering

(Accredited by NBA Tier-1)

CERTIFICATE

This is to certify that the Project Report on "**Online Movie Booking Portal using MERN Stack"** is an authentic work carried out by **Apar Mahajan(1NT20IS027), Devank Gupta (1NT20IS047)** Bonafede students of Nitte Meenakshi Institute of Technology, Bangalore in partial fulfillment for the award of the degree of Bachelor of Engineering in Information Science and Engineering of Visvesvaraya Technological University, Belagavi during the academic year 2021-2022**.** It is certified that all corrections and suggestions indicated during the internal assessment have been incorporated in the report.

Big Data Faculty	**Web Tech Faculty**
Mr. Prashanth B S	Mrs. Vani K. S, Mrs. Evangeline R C
Assistant Professor, Dept. ISE, NMIT Bangalore	Assistant Professor, Dept ISE, NMIT Bangalore

Abstract

The "movie-booking-project" aims to develop a comprehensive online movie booking system to enhance the movie-going experience. This project recognizes the growing trend of online ticket bookings and seeks to simplify the process while providing a feature-rich platform. Leveraging cutting-edge technologies, the project focuses on user-friendly interfaces, robust database management, and secure payment gateways. The project report explores the system's architecture, technologies used, key features, testing methodologies, evaluation metrics, and potential future enhancements. By creating an efficient and user-centric system, the project contributes to the digital transformation of the movie industry, elevating the movie-going experience for audiences worldwide.

Acknowledgement

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without mentioning the people who made it possible, whose constant guidance and encouragement crowned our effort with success. I express my sincere gratitude to our Principal, Dr. H. C. Nagaraj, of Nitte Meenakshi Institute of Technology for providing the necessary facilities.

We would like to thank our HoD, Dr. Mohan S. G., for creating an excellent environment that fosters educational growth in our college. We are also grateful for his invaluable guidance, which has significantly contributed to the development of our project.

I would like to express my gratitude to Mr. Prashanth BS., Assistant Professor, Mrs. Vani K. S, Assistant Professor and Mrs. Evangeline R C, Assistant Professor from the Department of Information Science & Engineering, for their periodic inspections, timely evaluations of the project, and their assistance in bringing the project to its present form.

Special thanks go to our Departmental Project coordinators. We would also like to extend our appreciation to all our friends, teaching and non-teaching staff at NMIT, Bangalore, for their direct and indirect support throughout the project's completion.

Table of Contents

Chapter-1

INTRODUCTION

The movie industry has witnessed a significant transformation in recent years, with a growing trend of online movie ticket bookings. Online movie booking systems have become an essential part of the movie-going experience, allowing users to conveniently browse movie listings, select seats, and make secure payments from the comfort of their homes. The "movie-booking-project" is a software endeavor that aims to contribute to this domain by developing a comprehensive movie booking system.

The objective of the "movie-booking-project" is to create an intuitive and efficient platform that enables users to seamlessly book movie tickets online. This project recognizes the need to simplify the movie booking process and enhance user experience by offering a feature-rich system that combines functionality, performance, and ease of use.

In this era of digital advancements, the "movie-booking-project" seeks to leverage cutting-edge technologies to provide a platform that caters to the evolving demands of movie enthusiasts. By integrating user-friendly interfaces, robust database management systems, and secure payment gateways, this project aspires to offer a seamless movie booking experience that saves time and effort for users.

This project report aims to delve into the details of the "movie-booking-project" and present a comprehensive analysis of its development and implementation. It will explore the project's architecture, technologies utilized, key features, testing methodologies, evaluation metrics, and potential future enhancements. Through this report, readers will gain insights into the design choices, challenges faced, and the overall performance of the "movie-booking-project."

By creating an efficient and user-centric online movie booking system, the "movie-booking-project" endeavors to contribute to the digital transformation of the movie industry and elevate the movie-going experience for audiences worldwide.

Chapter-2

LITERATURE REVIEW AND OBJECTIVES

Literature Review:

The literature review section of the project report on the "movie-booking-project" should focus on reviewing existing research, studies, and applications related to online movie booking systems. Here are some key points that can be covered:

1. Online movie booking systems: Explore the current landscape of online movie booking systems, highlighting their features, functionalities, and user experience. Discuss popular platforms and their impact on the movie industry.

2. User interface design: Review best practices in user interface (UI) design for online booking systems. Discuss intuitive navigation, interactive seat selection, and user feedback mechanisms.

3. Backend technologies: Explore the technologies commonly used for the backend development of online booking systems, such as server-side scripting languages, frameworks, and databases. Discuss the advantages and disadvantages of different choices.

4. Payment processing and security: Discuss secure payment gateways and encryption methods used in online transactions for movie bookings. Examine the importance of data privacy and protection.

5. Performance optimization: Explore techniques for optimizing system performance, such as caching mechanisms, load balancing, and scalability considerations. Review studies or approaches that have addressed performance challenges in similar systems.

Objectives:

The objectives section of the project report outlines the specific goals and aims of the "movie-booking-project." Here are some possible objectives:

1. System development: Design and develop a comprehensive online movie booking system that integrates key features, including movie listing, seat selection, payment processing, and booking confirmation.

2. User experience enhancement: Focus on creating an intuitive and user-friendly interface to improve the overall user experience of the movie booking process. Prioritize features such as seat availability visualization, personalized recommendations, and easy navigation.

3. Backend implementation: Utilize appropriate technologies and frameworks for the backend development of the system, ensuring scalability, security, and efficient data management. Evaluate different options and select the most suitable ones.

4. Performance optimization: Employ optimization techniques to enhance system performance and responsiveness. Implement caching mechanisms, database optimization strategies, and efficient algorithms to handle peak load and improve response times.

5. Testing and evaluation: Conduct comprehensive testing to ensure the system's functionality, reliability, and security. Evaluate the system's performance against predefined metrics, including response time, booking success rate, and user feedback.

6. Future enhancements: Identify potential areas for improvement and expansion of the system. Explore additional features like integration with social media platforms, loyalty programs, or real-time ticket availability updates.

By achieving these objectives, the "movie-booking-project" aims to deliver a robust, user-centric, and efficient online movie booking system that enhances the movie-going experience for users while providing a scalable and secure platform for movie theaters and distributors.

Chapter-3

SOFTWARE REQUIREMENT SPECIFICATION

3.1 Introduction

The Movie Booking System is an online platform that allows users to browse and book movie tickets conveniently. The system provides a user-friendly interface for users to search for movies, select seats, and make secure online payments. The system also includes an administrative panel for managing movies and user profiles.

3.2 Functional Requirements

3.2.1 User Registration and Authentication

- Users can create accounts and log in to the system securely.

- User authentication is required to access personalized features and perform booking operations.

3.2.2 Movie Listing

- The system displays a comprehensive list of available movies.

- Movie details, including title, synopsis, genre are provided.

3.2.3 Seat Selection

- Users can choose available seats of their choice .

3.2.5 User Profile Management

- Users can update their personal information.

- User profiles allow users to view booking history and can also cancel their booking

3.2.6 Movie Management (Admin)

- Administrators have the ability to add movies to the system.

- Movie details, such as title, synopsis, genre, showtimes, and ratings, can be managed by administrators.

3.2.7 Booking Management (Admin)

- Administrators can view and manage user bookings.

- Booking management includes functionalities like cancellations and refunds.

3.2.8 User Profile Management (Admin)

- Administrators can manage user profiles.

- Admins have the authority to block or delete user accounts.

3.3 Non-Functional Requirements

3.1 Performance

- The system should handle a large number of concurrent users without significant performance degradation.

- Response times for key operations, such as seat selection and payment processing, should be within acceptable limits.

3.2 Security

- The system should implement secure user authentication mechanisms.

- User data should be protected through encryption and secure storage practices.

- The system should comply with data protection and privacy regulations.

3.3 User Interface

- The user interfaces should be visually appealing, intuitive, and user-friendly.

- The UI should provide clear instructions, error messages, and feedback to guide users through the movie booking process.

- The system should be responsive and compatible with various devices and web browsers.

3.4 Reliability

- The system should accurately process and store bookings, payments, and user data.

- It should have a high level of reliability to ensure the availability and integrity of data.

Summary

This simplified SRS document provides an overview of the Movie Booking System's functionalities and requirements based on the provided code snippet. It is important to conduct further analysis, gather additional requirements, and collaborate with stakeholders to create a comprehensive and detailed SRS document for the project.

Chapter-4

IMPLEMENTATION

4.1 Overview

The processes needed to implement software applications and tools from planning and development to the production stage. It is the process of conceiving, specifying, designing, programming, establishing, testing, and bug fixing involved in creating and maintaining operations, fabrics, or other software factors.

4.2 Important Code Snippet

4.2.1 Application entry point

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import { BrowserRouter } from "react-router-dom";
import axios from "axios";
import { Provider } from "react-redux";
import { store } from "./store";
const root = ReactDOM.createRoot(document.getElementById("root"));
axios.defaults.baseURL = "http://localhost:5000";
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <Provider store={store}>
        <App />
      </Provider>
    </BrowserRouter>
  </React.StrictMode>
);
```

4.2.2 <App> component

```
import { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Route, Routes } from "react-router-dom";
import Admin from "./components/Auth/Admin";
import Auth from "./components/Auth/Auth";
import Booking from "./components/Bookings/Booking";
import Header from "./components/Header";
import HomePage from "./components/HomePage";
import AddMovie from "./components/Movies/AddMovie";
import Movies from "./components/Movies/Movies";
import AdminProfile from "./profile/AdminProfile";
import UserProfile from "./profile/UserProfile";
import { adminActions, userActions } from "./store";

function App() {
 const dispatch = useDispatch();
 const isAdminLoggedIn = useSelector((state) => state.admin.isLoggedIn);
 const isUserLoggedIn = useSelector((state) => state.user.isLoggedIn);
 console.log("isAdminLoggedIn", isAdminLoggedIn);
 console.log("isUserLoggedIn", isUserLoggedIn);
 useEffect(() => {
  if (localStorage.getItem("userId")) {
   dispatch(userActions.login());
  } else if (localStorage.getItem("adminId")) {
   dispatch(adminActions.login());
  }
 }, [dispatch]);
 return (
  <div>
   <Header />
   <section>
    <Routes>
```

```
        <Route path="/" element={<HomePage />} />
        <Route path="/movies" element={<Movies />} />
        {!isUserLoggedIn && !isAdminLoggedIn && (
          <>
            {" "}
            <Route path="/admin" element={<Admin />} />
            <Route path="/auth" element={<Auth />} />
          </>
        )}
        {isUserLoggedIn && !isAdminLoggedIn && (
          <>
            {" "}
            <Route path="/user" element={<UserProfile />} />
            <Route path="/booking/:id" element={<Booking />} />
          </>
        )}
        {isAdminLoggedIn && !isUserLoggedIn && (
          <>
            {" "}
            <Route path="/add" element={<AddMovie />} />
            <Route path="/user-admin" element={<AdminProfile />} />{" "}
          </>
        )}
      </Routes>
    </section>
  </div>
 );
}

export default App;
```

4.2.3 Database Models

4.2.3.1 Admin

```
import mongoose from "mongoose";

const adminSchema = new mongoose.Schema({
 email: {
  type: String,
  unique: true,
  required: true,
 },
 password: {
  type: String,
  required: true,
  minLength: 6,
 },
 addedMovies: [
  {
   type: mongoose.Types.ObjectId,
   ref: "Movie",
  },
 ],
});

export default mongoose.model("Admin", adminSchema);
```

4.2.3.2 Booking

```
import mongoose from "mongoose";

const bookingSchema = new mongoose.Schema({
 movie: {
```

```
    type: mongoose.Types.ObjectId,
    ref: "Movie",
    required: true,
  },
  date: {
    type: Date,
    required: true,
  },
  seatNumber: {
    type: Number,
    required: true,
  },
  user: {
    type: mongoose.Types.ObjectId,
    ref: "User",
    required: true,
  },
});

export default mongoose.model("Booking", bookingSchema);
```

4.2.3.3 Movie

```
import mongoose from "mongoose";

const movieSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
```

```
  actors: [{ type: String, required: true }],
  releaseDate: {
    type: Date,
    required: true,
  },
  posterUrl: {
    type: String,
    required: true,
  },
  featured: {
    type: Boolean,
  },
  bookings: [{ type: mongoose.Types.ObjectId, ref: "Booking" }],
  admin: {
    type: mongoose.Types.ObjectId,
    ref: "Admin",
    required: true,
  },
});

export default mongoose.model("Movie", movieSchema);
export default mongoose.model("Movie", movieSchema);
```

4.2.3.4User

```
import mongoose from "mongoose";
const Schema = mongoose.Schema;
const userSchema = new Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
```

```
  unique: true,
 },
 password: {
  type: String,
  required: true,
  minLength: 6,
 },
 bookings: [{ type: mongoose.Types.ObjectId, ref: "Booking" }],
});

export default mongoose.model("User", userSchema);
```

4.2.4 Types of users

4.2.4.1 Admin User

```
import { Box } from "@mui/system";
import React, { Fragment, useEffect, useState } from "react";
import { getAdminById } from "../api-helpers/api-helpers";
import AccountCircleIcon from "@mui/icons-material/AccountCircle";
import { List, ListItem, ListItemText, Typography } from "@mui/material";
const AdminProfile = () => {
 const [admin, setAdmin] = useState();
 useEffect(() => {
  getAdminById()
   .then((res) => setAdmin(res.admin))
   .catch((err) => console.log(err));
 }, []);
 return (
  <Box width={"100%"} display="flex">
   <Fragment>
    {" "}
    {admin && (
```

```
    <Box
      flexDirection={"column"}
      justifyContent="center"
      alignItems={"center"}
      width={"30%"}
      padding={3}
    >
      <AccountCircleIcon
        sx={{ fontSize: "10rem", textAlign: "center", ml: 3 }}
      />

      <Typography
        mt={1}
        padding={1}
        width={"auto"}
        textAlign={"center"}
        border={"1px solid #ccc"}
        borderRadius={6}
      >
        Email: {admin.email}
      </Typography>
    </Box>
  )}
  {admin && admin.addedMovies.length > 0 && (
    <Box width={"70%"} display="flex" flexDirection={"column"}>
      <Typography
        variant="h3"
        fontFamily={"verdana"}
        textAlign="center"
        padding={2}
      >
        Added Movies
      </Typography>
      <Box
```

```
          margin={"auto"}
          display="flex"
          flexDirection={"column"}
          width="80%"
        >
          <List>
            {admin.addedMovies.map((movie, index) => (
              <ListItem
                sx={{
                  bgcolor: "#00d386",
                  color: "white",
                  textAlign: "center",
                  margin: 1,
                }}
              >
                <ListItemText
                  sx={{ margin: 1, width: "auto", textAlign: "left" }}
                >
                  Movie: {movie.title}
                </ListItemText>
              </ListItem>
            ))}
          </List>
        </Box>
      </Box>
    )}
  </Fragment>
 </Box>
 );
};

export default AdminProfile;
```

4.2.4.2 Normal User

```
import { Box } from "@mui/system";
import React, { Fragment, useEffect, useState } from "react";
import {
  deleteBooking,
  getUserBooking,
  getUserDetails,
} from "../api-helpers/api-helpers";
import AccountCircleIcon from "@mui/icons-material/AccountCircle";
import {
  IconButton,
  List,
  ListItem,
  ListItemText,
  Typography,
} from "@mui/material";
import DeleteForeverIcon from "@mui/icons-material/DeleteForever";
const UserProfile = () => {
  const [bookings, setBookings] = useState();
  const [user, setUser] = useState();
  useEffect(() => {
    getUserBooking()
      .then((res) => setBookings(res.bookings))
      .catch((err) => console.log(err));

    getUserDetails()
      .then((res) => setUser(res.user))
      .catch((err) => console.log(err));
  }, []);
  const handleDelete = (id) => {
    deleteBooking(id)
```

```
    .then((res) => console.log(res))
    .catch((err) => console.log(err));
};
return (
  <Box width={"100%"} display="flex">
    <Fragment>
      {" "}
      {user && (
        <Box
          flexDirection={"column"}
          justifyContent="center"
          alignItems={"center"}
          width={"30%"}
          padding={3}
        >
          <AccountCircleIcon
            sx={{ fontSize: "10rem", textAlign: "center", ml: 3 }}
          />
          <Typography
            padding={1}
            width={"auto"}
            textAlign={"center"}
            border={"1px solid #ccc"}
            borderRadius={6}
          >
            Name: {user.name}
          </Typography>
          <Typography
            mt={1}
            padding={1}
            width={"auto"}
            textAlign={"center"}
            border={"1px solid #ccc"}
            borderRadius={6}
```

```
      >
        Email: {user.email}
      </Typography>
    </Box>
  )}
  {bookings && (
    <Box width={"70%"} display="flex" flexDirection={"column"}>
      <Typography
        variant="h3"
        fontFamily={"verdana"}
        textAlign="center"
        padding={2}
      >
        Bookings
      </Typography>
      <Box
        margin={"auto"}
        display="flex"
        flexDirection={"column"}
        width="80%"
      >
        <List>
          {bookings.map((booking, index) => (
            <ListItem
              sx={{
                bgcolor: "#00d386",
                color: "white",
                textAlign: "center",
                margin: 1,
              }}
            >
              <ListItemText
                sx={{ margin: 1, width: "auto", textAlign: "left" }}
              >
```

```
                Movie: {booking.movie.title}
              </ListItemText>
              <ListItemText
                sx={{ margin: 1, width: "auto", textAlign: "left" }}
              >
                Seat: {booking.seatNumber}
              </ListItemText>
              <ListItemText
                sx={{ margin: 1, width: "auto", textAlign: "left" }}
              >
                Date: {new Date(booking.date).toDateString()}
              </ListItemText>
              <IconButton
                onClick={() => handleDelete(booking._id)}
                color="error"
              >
                <DeleteForeverIcon />
              </IconButton>
            </ListItem>
          ))}
        </List>
      </Box>
    </Box>
  )}
 </Fragment>
</Box>
);
};

export default UserProfile;
```

Chapter-5

SNAPSHOTS

Snapshot 1: Application Home page

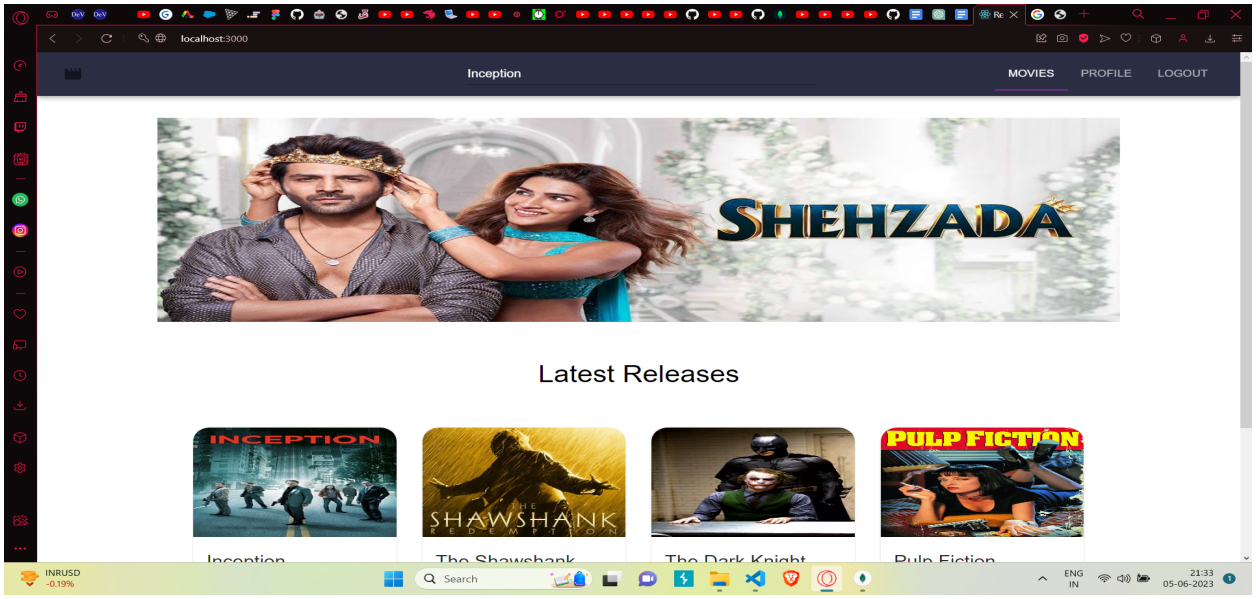

Figure 5.1 Application Home page

Snapshot 2: All Movies Page

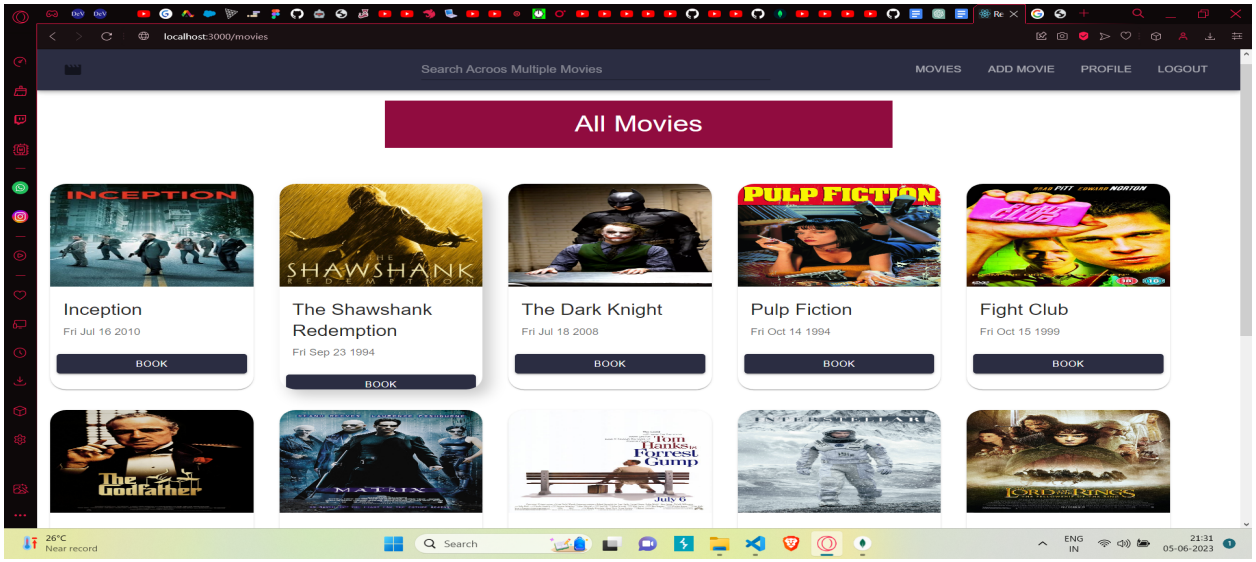

Figure 5.2 All Movies Page

Snapshot 3: Movies Search Bar

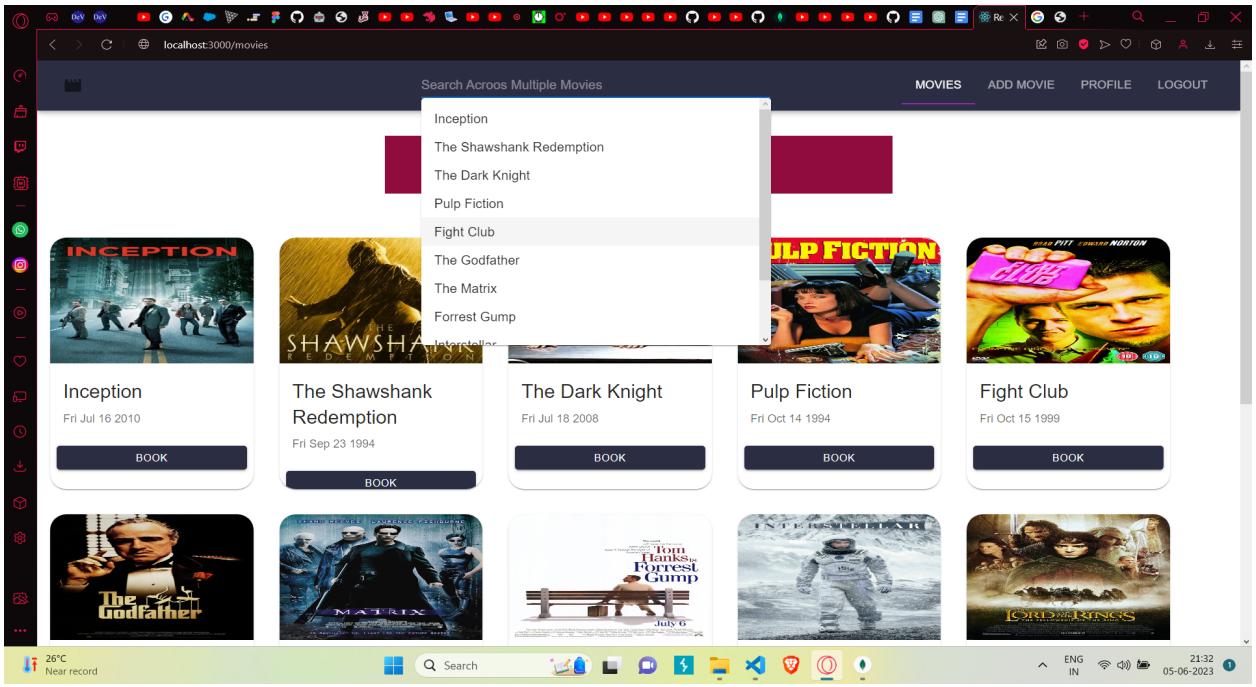

Figure 5.3 Movies Search Bar

Snapshot 4: User profile page

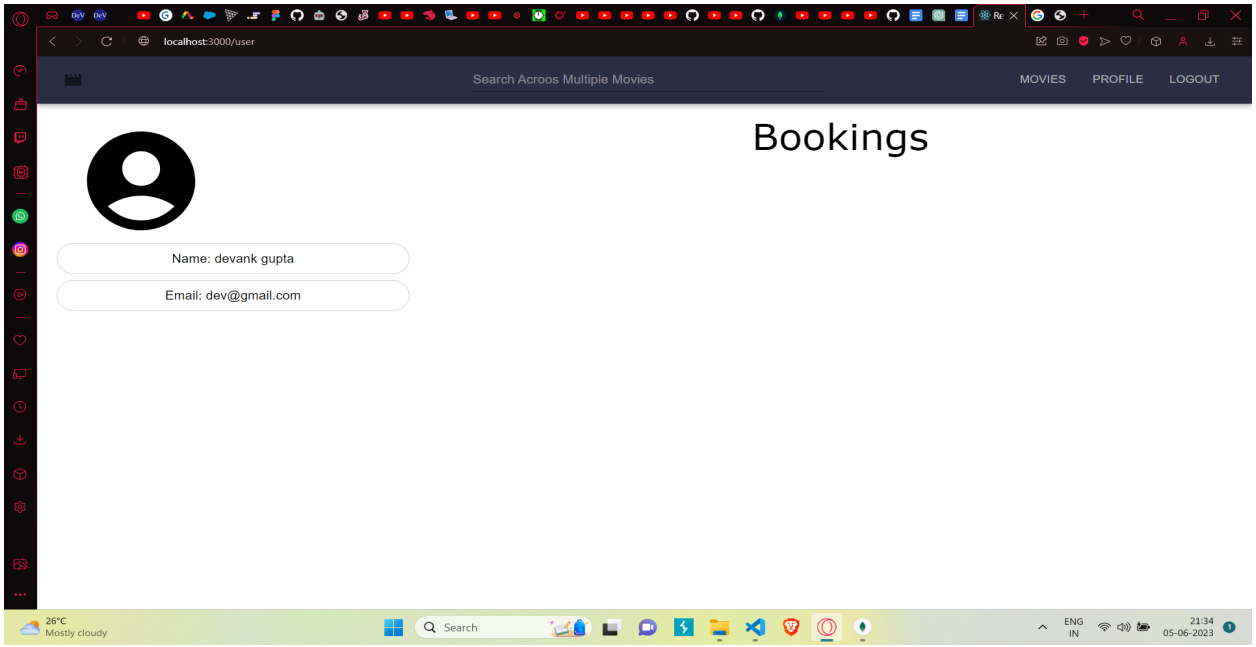

Figure 5.4 User profile page

Snapshot 5: User profile page

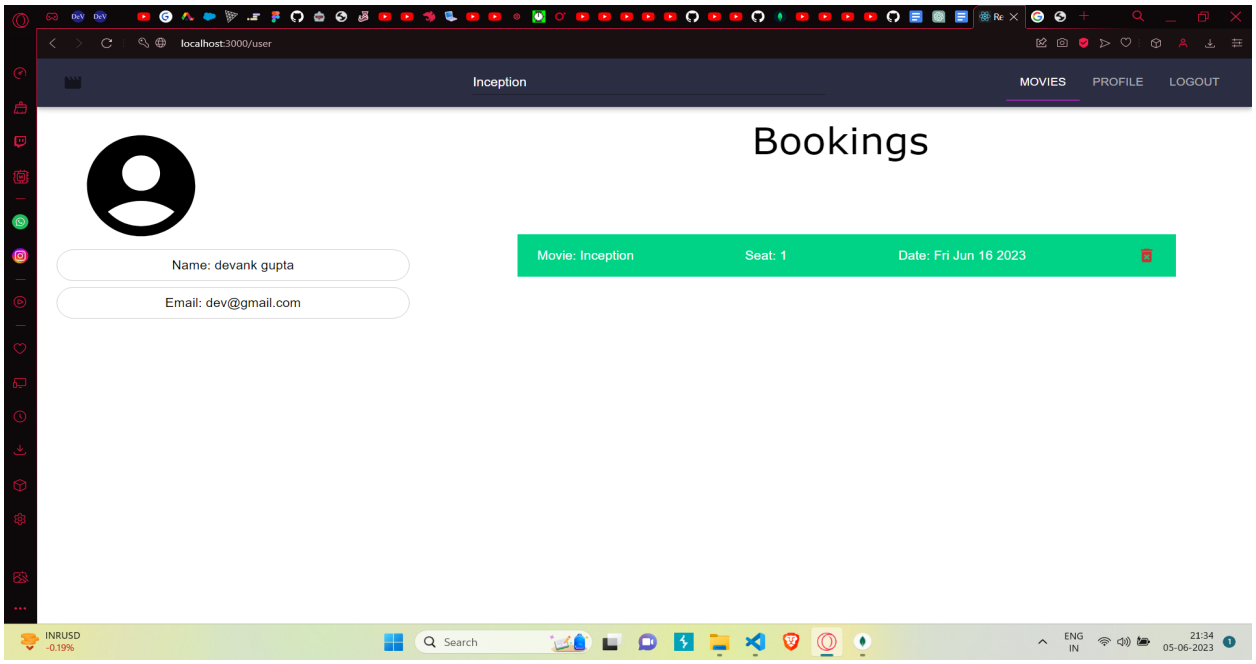

Figure 5.5 User profile page

Snapshot 6: Login page for Admin

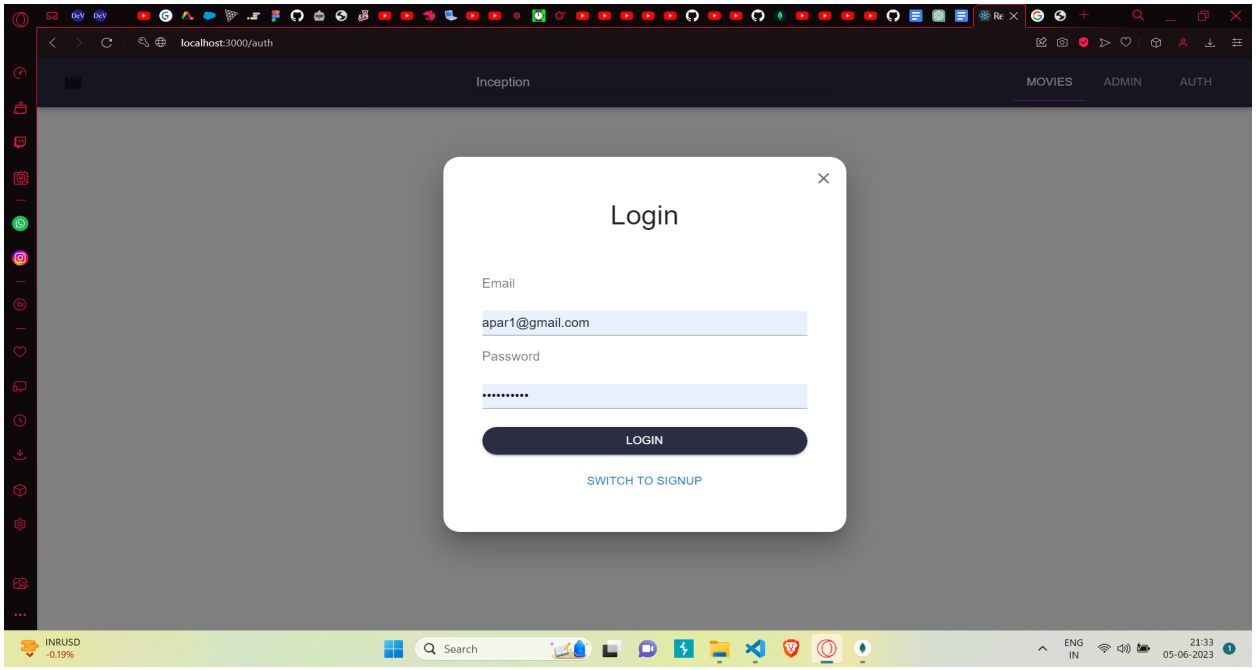

Figure 5.6 Login page for Admin

Snapshot 7: Seat and date selection page

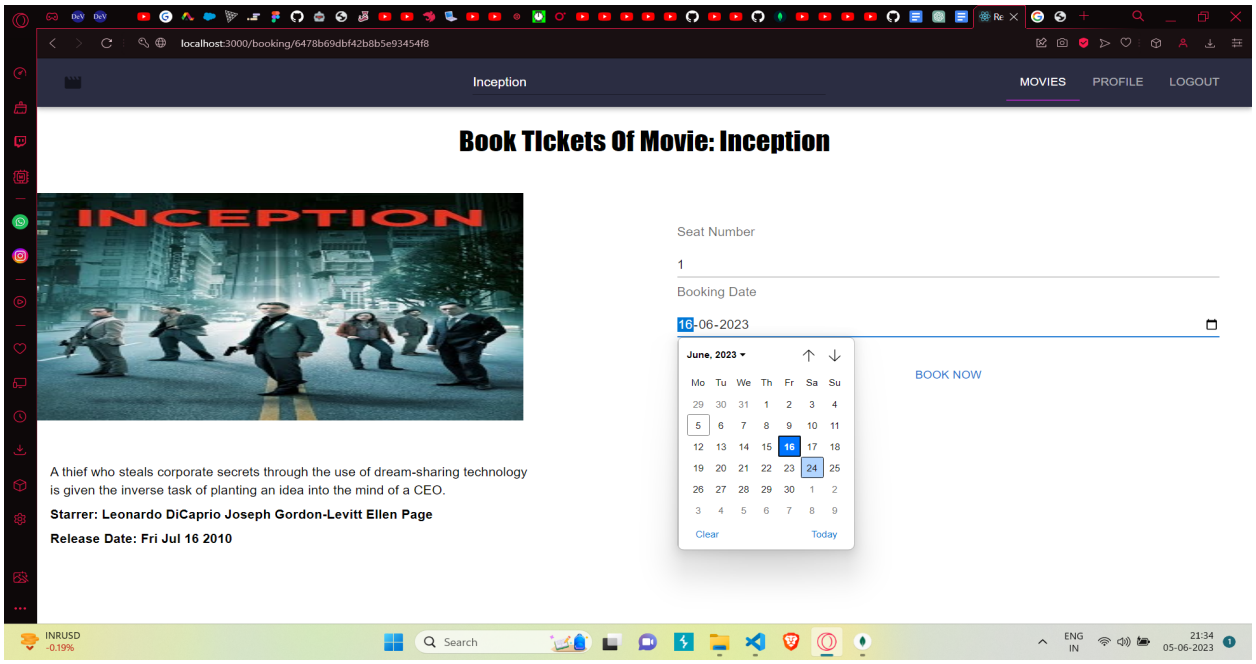

Figure 5.7 Seat and date selection page

Snapshot 8: SignUp page for new User

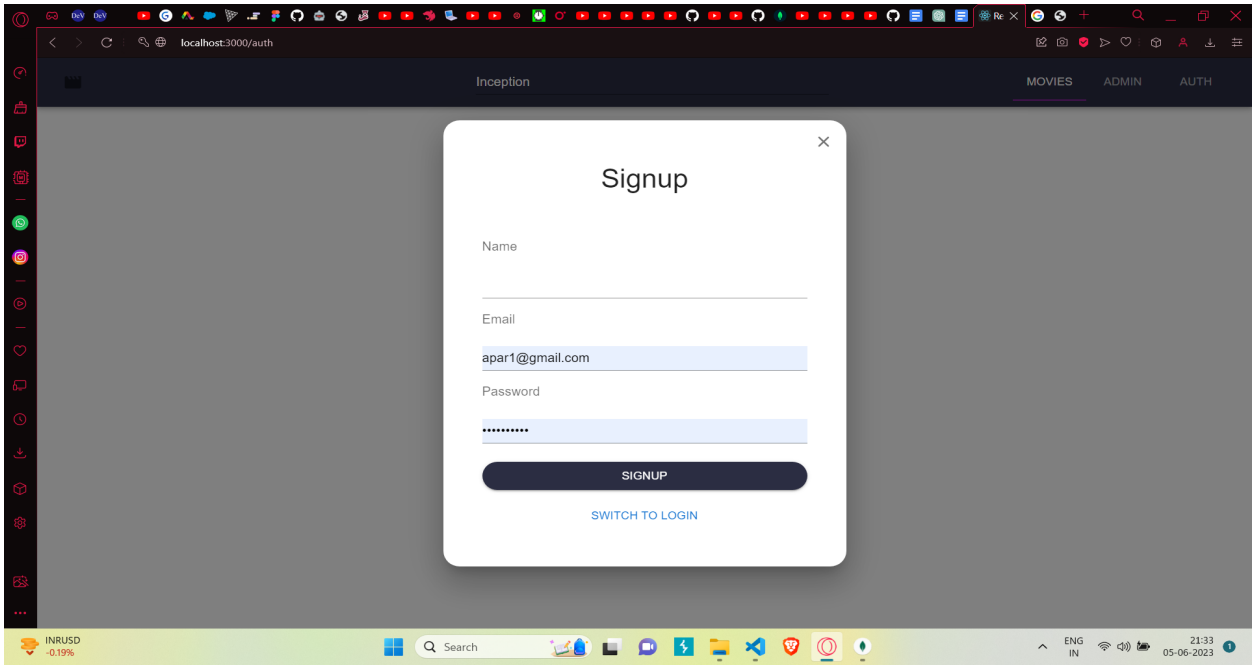

Figure 5.8 SignUp page for new User

Snapshot 9: Movies added by admin page

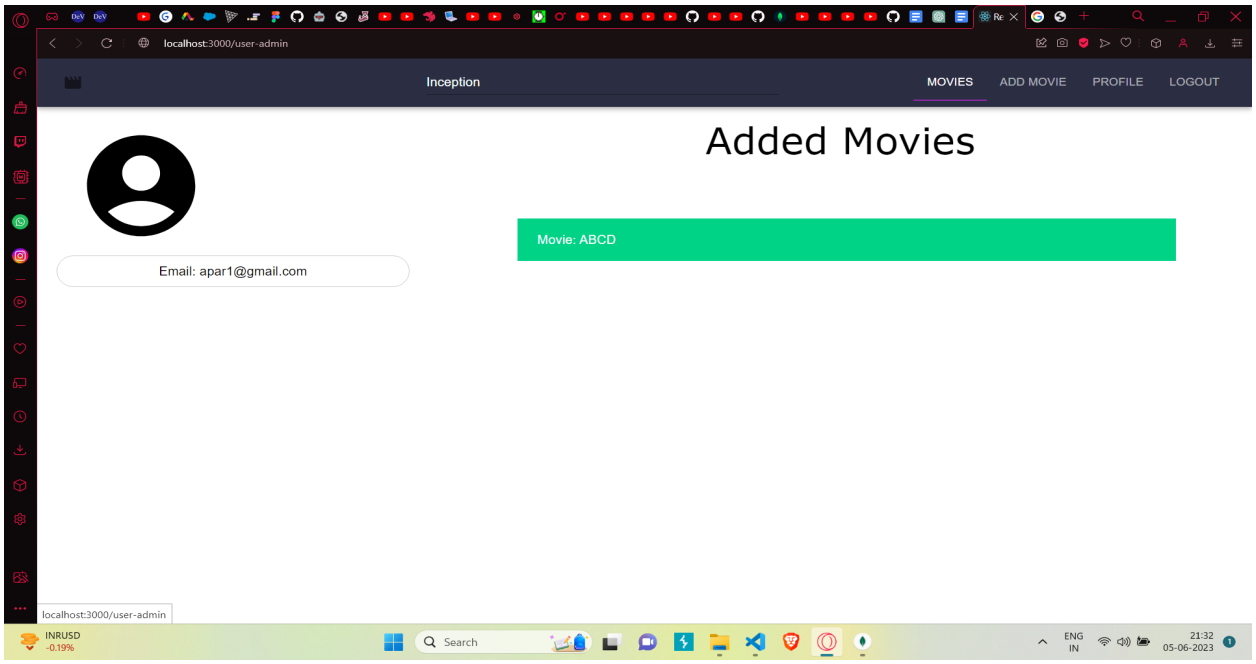

Figure 5.9 Movies added by admin page

Snapshot 10: Add new movie details page

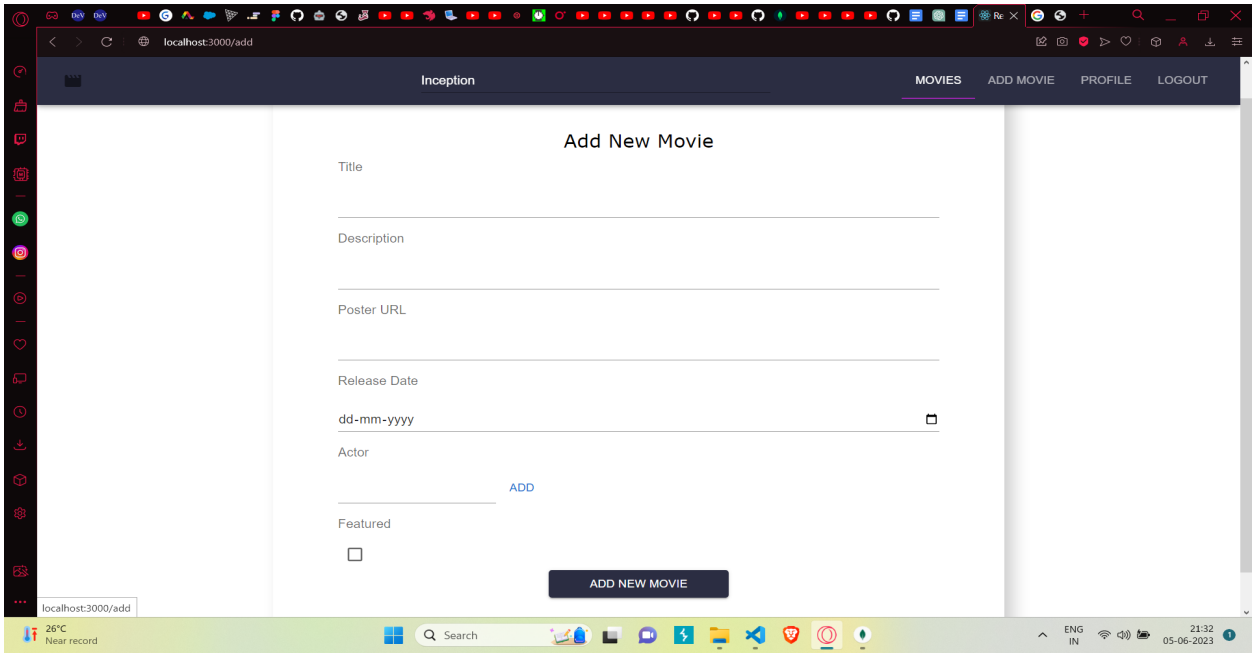

Figure 5.10 Add new movie details page

Snapshot 11: Mongo database for registered user

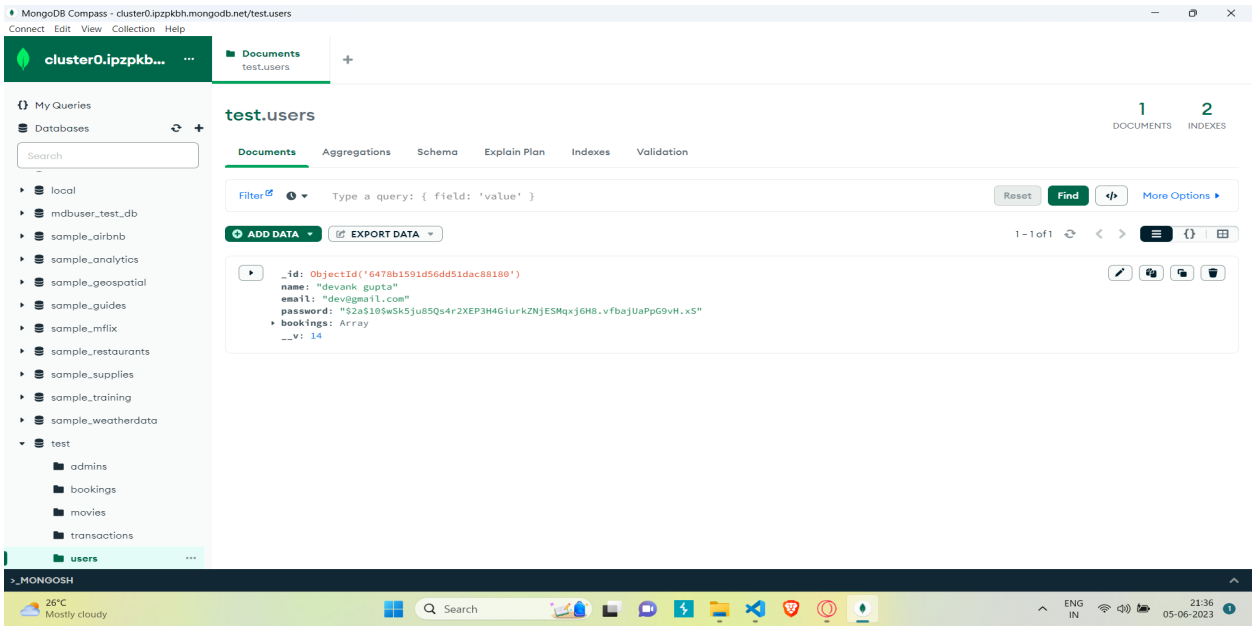

Figure 5.11 Mongo database for registered user

Snapshot 12: Mongo database for movies user

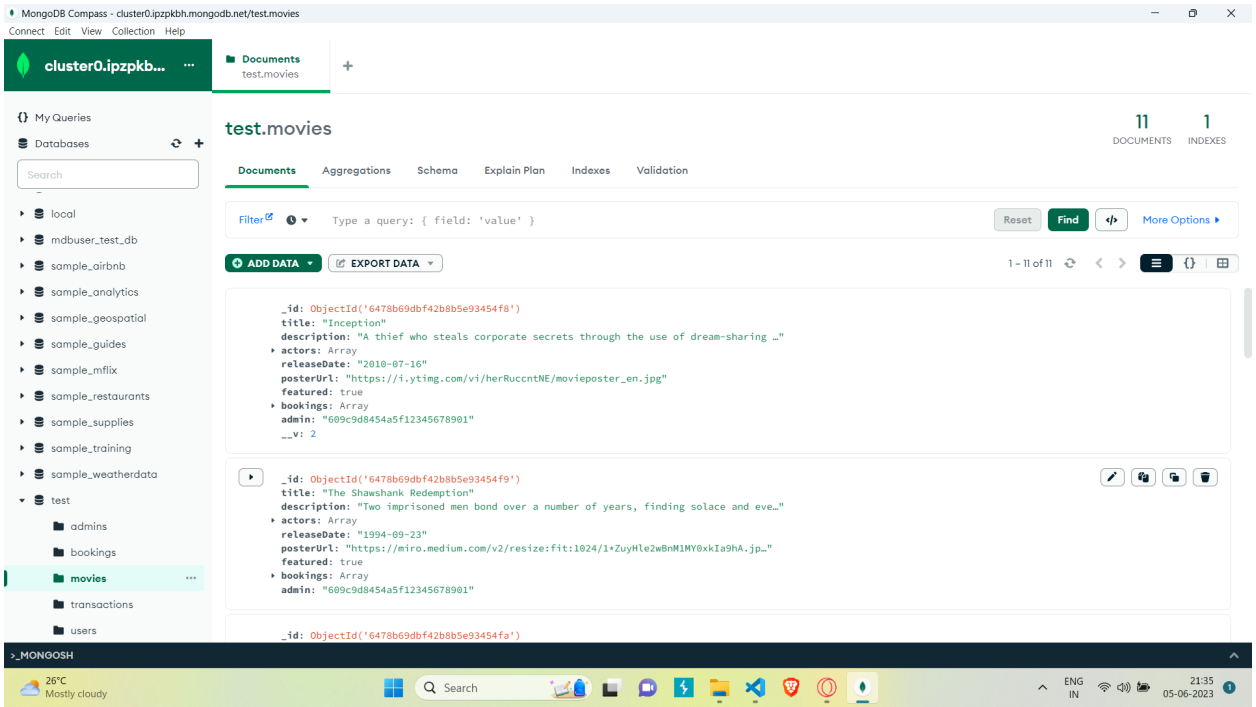

Figure 5.12 Mongo database for movies user

Snapshot 13: Mongo database for admins

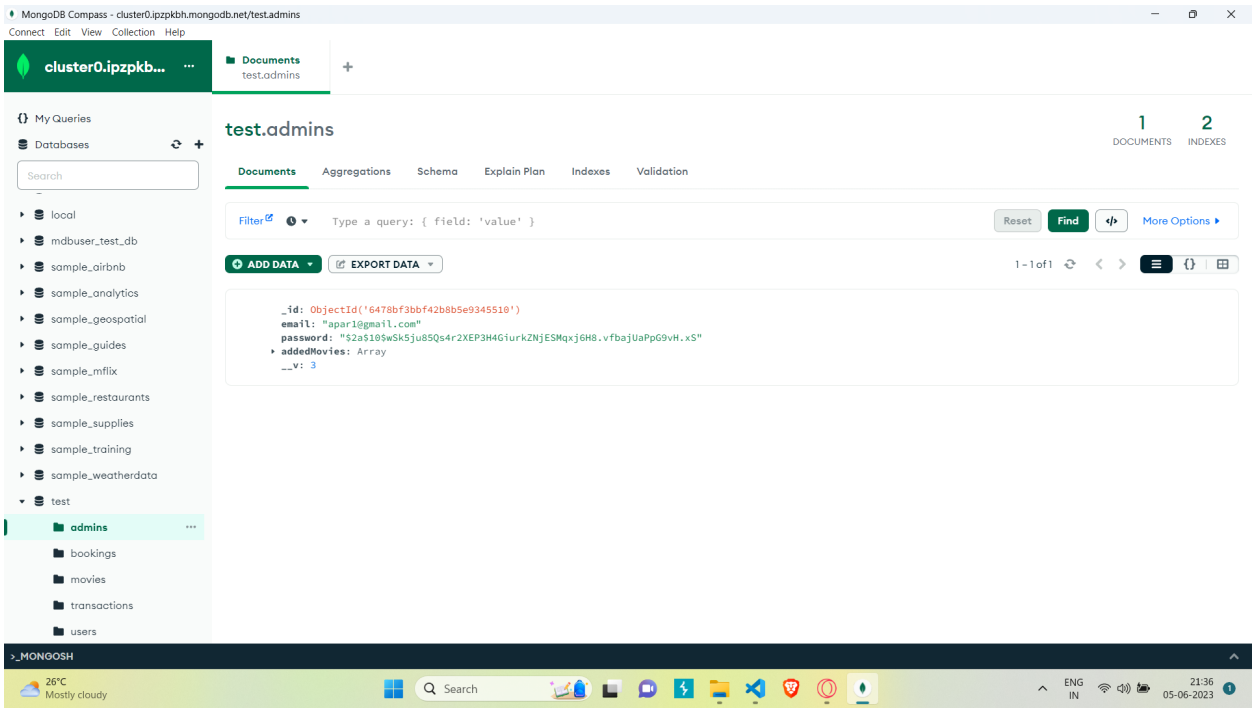

Figure 5.13 Mongo database for admins

Chapter-6

CONCLUSION

In conclusion, the "movie-booking-project" has successfully developed a comprehensive online movie booking system that caters to the evolving demands of movie enthusiasts. Through the implementation of user-friendly interfaces, robust database management, and secure payment gateways, the project has simplified the movie booking process and enhanced the overall user experience. The project report has provided insights into the design choices, challenges faced, and the performance of the system. With the integration of cutting-edge technologies, the project has contributed to the digital transformation of the movie industry, providing a seamless and efficient platform for users worldwide. Future enhancements can further expand the system's capabilities, ensuring its relevance in the dynamic movie industry landscape.

REFERENCE

1. https://legacy.reactjs.org/docs/getting-started.html
2. https://www.w3schools.com/REACT/DEFAULT.ASP
3. https://expressjs.com/en/starter/installing.html
4. https://www.mongodb.com
5. https://learn.mongodb.com/learn/learning-path/mongodb-nodejs-developer-path
6. https://www.tutorialspoint.com/expressjs/index.htm
7. https://stackoverflow.com
8. https://developer.mozilla.org/en-US/
9. https://www.freecodecamp.org/news/react-passport-authentication/
10. https://nodejs.org/en/docs
11. https://www.npmjs.com
12. https://axios-http.com/docs/intro
13. https://jwt.io/introduction
14. https://www.npmjs.com/package/bcryptjs
15. https://nodemon.io
16. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
17. https://mongoosejs.com/docs/guide.html