Movies And Reviews System

Submitted for the course

SmartBridge

Modern Application Development (Java Spring Boot)

Submitted By

20BCT0202 - Naman Rishi

20BCE2124 - Aryan Sharma

20BCE2322 - Rachit Saxena

20BKT0093 - Anuranan Goswami

CONTENTS

1. Introduction

1.1. Overview

The project aims to create a platform for movie enthusiasts to explore the latest movies, share reviews, and engage in discussions. Users can discover upcoming movies, post their opinions, and check reviews from others. The application enhances the movie-watching experience, promotes community engagement, and facilitates informed decision-making.

It follows a loosely coupled architecture, combining Java Spring Boot for the backend API and React for the frontend web application. MongoDB serves as the database to store movie data and user reviews. Future deployment using Docker and Kubernetes ensures scalability and reliability.

1.2. Purpose

The purpose of this project is to offer a platform for movie fans to discover, discuss, and share their opinions on movies. Users can explore the most recent and upcoming films and express opinions, ratings, and reviews on films they have seen. The project supports viewers in making informed decisions about which films to watch by combining user reviews and ratings. This software can help with a community-driven approach to movie recommendations and discussions.

The project aims to bring together movie enthusiasts, providing them with a platform to connect, interact, and engage in discussions about movies. Users can rely on the collective knowledge of the community to assess the acceptability of a film based on the feedback and views of others.

2. Literature Survey

No.	Title	Author	Proposed Work
1.	Modern API Development with Spring and Spring Boot: Design highly scalable and maintainable APIs with REST, gRPC, GraphQL, and the reactive paradigm (2021)	S. Sharma	This book is a comprehensive guide that explores the fundamentals of RESTful APIs, Spring and Spring Boot concepts, API specifications, business logic implementation, and asynchronous API design. The book provides practical insights, best practices, and hands-on examples for building scalable and efficient APIs. It covers a range of topics including API design principles, error handling, data persistence, and the migration to asynchronous programming. This book serves as a valuable resource for developers seeking to enhance their API development skills using Spring and Spring Boot frameworks.

2.	Beginning React: Simplify your frontend development workflow and enhance the user experience of your applications with React (2018)	A. Chiarelli	This book introduces React and covers key aspects of frontend development using React. The book provides insights into designing user interfaces, creating and managing components, and handling user interactivity. With practical examples and a focus on enhancing the user experience, this book serves as a valuable resource for developers seeking to streamline their frontend development workflow and leverage the power of React in their applications.
3.	Comparison of MySQL and MongoDB with focus on performance (2020)	P. Filip, L. Čegan	The paper provides an overview and comparison of NoSQL databases, specifically focusing on the performance differences between MySQL and MongoDB. The study includes benchmark analyses of various operations, such as data insertion, update, and deletion, with and without transactions. The paper aims to evaluate the impact of data storage structures on database performance and assess the added value of benchmarks in terms of indexed field costs. This research contributes to the understanding of performance considerations in selecting between MySQL and MongoDB for different application scenarios.
4.	Light-Weight and Scalable Hierarchical-MVC Architecture for Cloud Web Applications (2018)	M. Ma, J. Yang, P. Wang, W. Liu J. Zhang	This paper addresses the challenges of modular and scalable web application development in the cloud computing era. The proposed approach, called Web Module Definition (WMD), introduces a hierarchical-MVC architecture that supports feature-based modularization and application structure. By decomposing the web application into interconnected modules, WMD enables better scalability and maintainability. The paper presents a demonstration website and a web application framework implementation that supports the WMD-based architecture. This research contributes to the advancement of web application development in the cloud by offering a lightweight and scalable solution for modularizing and organizing web applications.

5.	Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform (2019)	J. Shah, D. Dubaria	The presented work explores the transformative impact of Docker and Kubernetes on cloud infrastructure and DevOps practices. The abstract highlights the capabilities of Docker in building, shipping, and running applications using containers, resulting in faster deployments, resource efficiency, and reliability. It also emphasizes the automation of container management, deployment, and scaling provided by Kubernetes, along with the benefits of using Google Cloud Platform for deploying containers on Kubernetes Engine. This paper serves as a concise resource for understanding the significance of Docker, Kubernetes, and Google Cloud Platform in developing modern cloud architectures and facilitating efficient application development and management.

2.1. Existing problem

The existing problem in the domain of movie platforms is the fragmentation of movie information and limited interactivity experienced on current movie websites or apps. For example, popular movie review websites such as IMDb, Rotten Tomatoes, and Metacritic provide valuable movie information and user reviews but lack seamless integration and a comprehensive user experience. Users often need to navigate between multiple platforms to gather information, read reviews, and engage with the movie community.

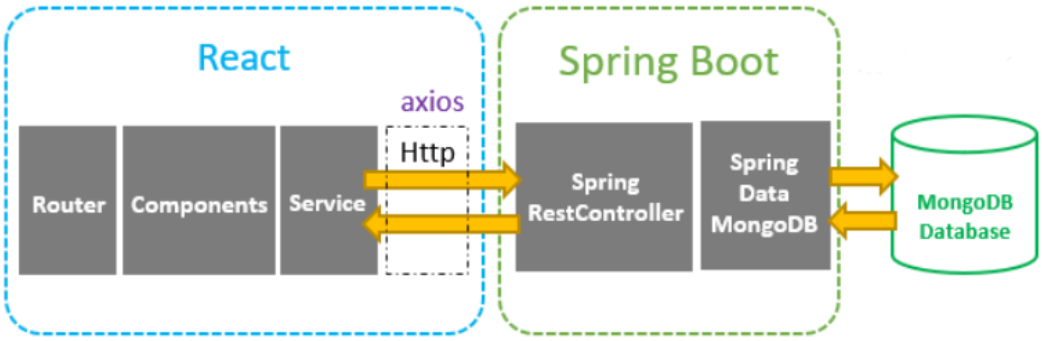
2.2. Proposed Solution

To address the existing problem, our project proposes the development of a comprehensive movies and reviews platform. Our solution aims to provide a centralized platform that offers the latest and upcoming movie listings, user-generated reviews, and interactive community features. By combining Java Spring Boot for the backend API, React for the frontend web application, and MongoDB as the database, our proposed solution provides a seamless and user-friendly experience.

Through our proposed solution, we aim to overcome the existing problem of fragmented movie information and limited interactivity by providing a centralized and engaging movies and reviews platform. Our goal is to create a user-centric and comprehensive environment where movie enthusiasts can access all the necessary information, share their thoughts, and actively participate in the movie community.

3. Theoretical Analysis

3.1. Block Diagram



3.2. Hardware / Software designing

The hardware requirements for our web application platform are as follows:

i. Server: We will require a server to host the backend API and the database. The server should have sufficient processing power and memory to handle the expected user traffic and database operations efficiently.

ii. Storage: Adequate storage space is necessary to store movie data and user reviews. We will utilize MongoDB as the database management system, so the server should have enough storage capacity to accommodate the growing database.

iii. Networking: A stable internet connection is essential to ensure uninterrupted access to the web application. The server should be connected to a reliable network infrastructure that can handle the expected traffic volume.

The software requirements for our web application platform are as follows:

i. JDK: Required for Java development.

ii. Maven: Used for build automation and dependency management.

iii. IntelliJ IDE: Provides a feature-rich development environment for Java.

iv. MongoDB Atlas: Cloud-based database service for storing movie data and reviews.

v. Bootstrap: CSS framework for creating a responsive and visually appealing UI.

vi. React JS: JavaScript library for building dynamic user interfaces.

vii. Axios: Simplifies frontend-backend communication with HTTP requests.

4. Experimental Investigations

Experimental investigations were conducted during the development of our platform to assess its functionality and performance. These investigations involved basic functionality testing and user experience evaluation. The following outline the key experiments and observations made:

i. Functionality Testing

To validate the basic functionality of our web application, we performed the following tests:

- Data Retrieval: We verified the successful fetching of movie data from the database and ensured its proper display on the frontend.
- Comment Posting: We tested the ability of users to post comments on movies and validated their storage in the database.
- Comment Viewing: We confirmed that users could view comments associated with each movie and that the comments were accurately displayed.

ii. Performance Testing

To assess the performance of our web application, we conducted tests to measure its responsiveness. Key performance metrics we evaluated included:

- Response Time: We measured the response time of the API endpoints to ensure prompt retrieval and display of movie data and comments.
- Scalability: We analysed how the system handled concurrent user requests and ensured that it remained responsive under moderate loads.

iii. User Experience Evaluation

We conducted user experience evaluations to gather feedback on the usability and intuitiveness of our web application. This involved:

- Usability Testing: We observed users interacting with the web application and recorded any usability issues or areas for improvement.

iv. Observations and Iterative Enhancements

Based on our experimental investigations, we made several observations and identified areas for improvement. These observations guided us in implementing iterative enhancements to enhance the basic functionality, performance, and user experience of our web application platform.

By conducting these experimental investigations, we ensured that our solution met the desired basic requirements, delivered a satisfactory user experience, and performed adequately under normal usage scenarios.

5. Flowchart

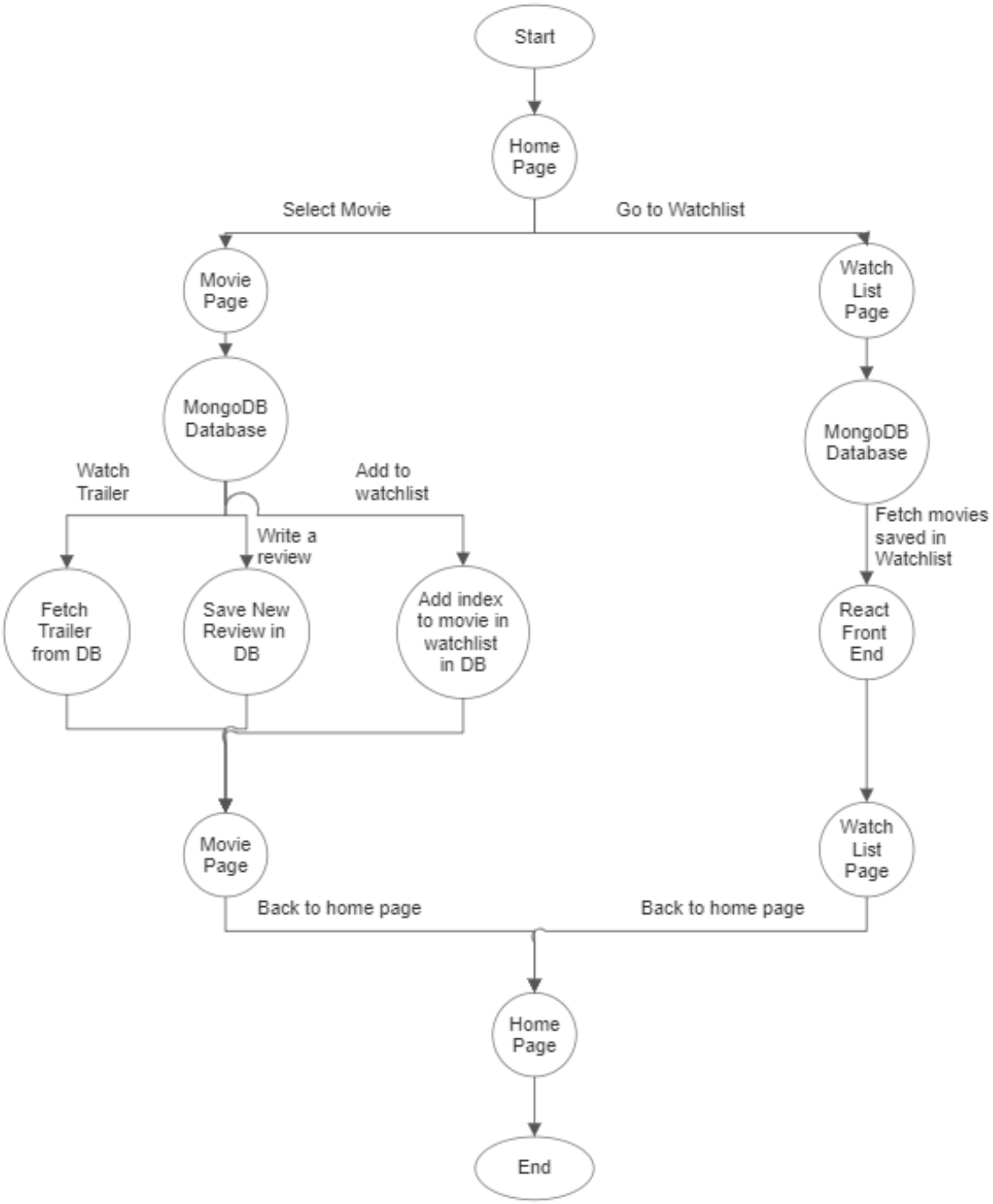
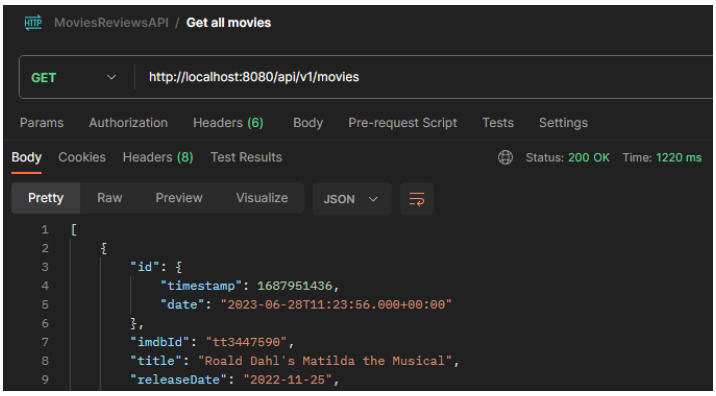
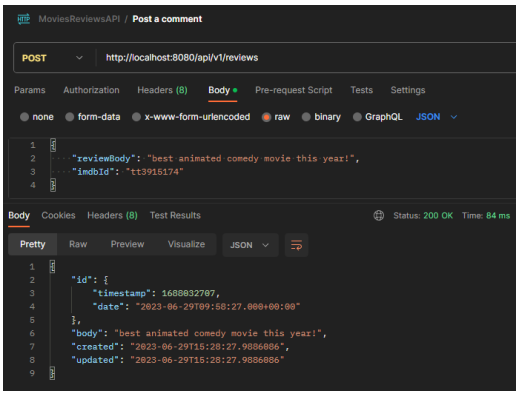


Fig.: Control Flow Graph

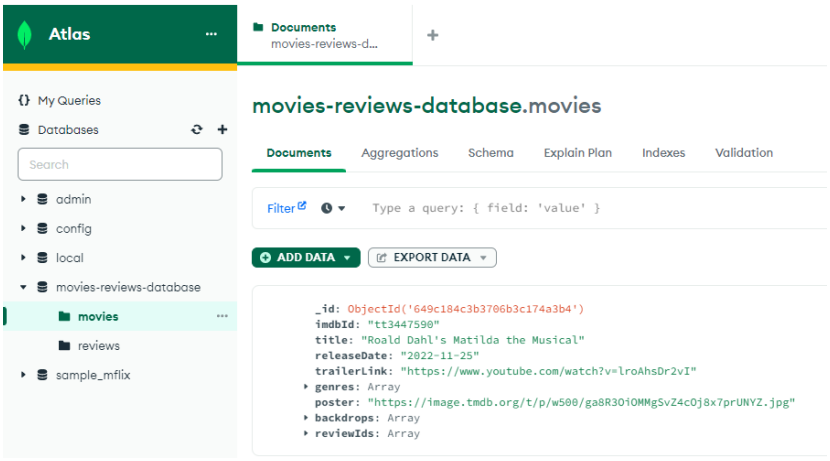
6. Result

For the backend, we successfully built a robust API using Java Spring Boot. This API serves as the foundation of our web application, allowing users to access movie data and post comments. It efficiently retrieves movie information from the database and handles user interactions, ensuring smooth functionality and seamless communication between the frontend and backend.

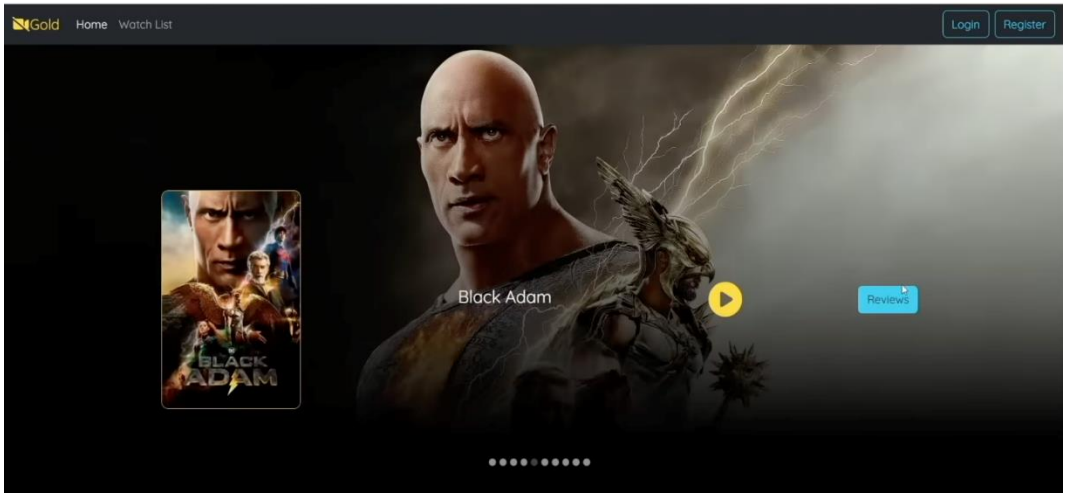


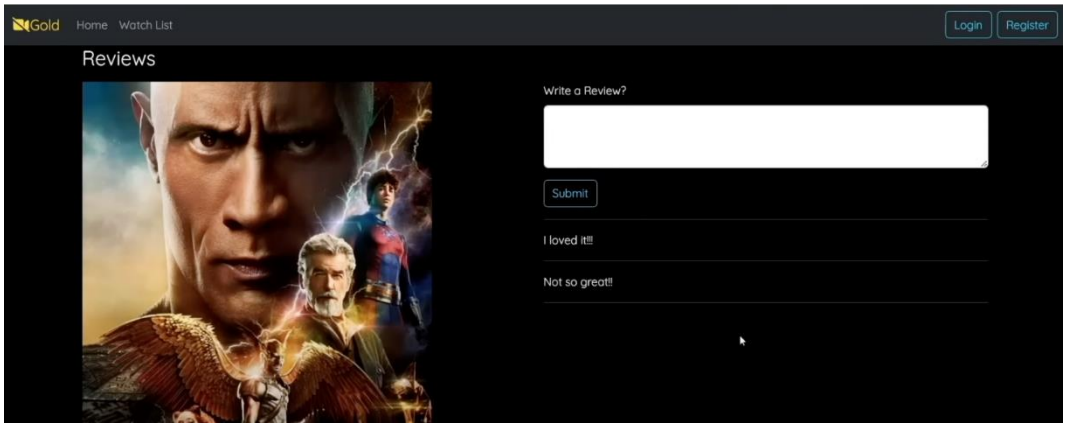


In terms of database integration, we implemented MongoDB Atlas as the backend database for storing movie data and user comments. The integration with MongoDB was seamless, providing reliable data storage and retrieval. The database effectively handles the storage and retrieval of movie information, ensuring the persistence of user-generated comments. This ensures that users' comments are accurately stored and readily available for viewing.



On the frontend, we developed a user-friendly web application using React. The frontend interface provides an intuitive and visually appealing experience for users. They can effortlessly read comments, post new comments, and view movie trailers. The React components were designed to offer a smooth and responsive user interface, enhancing the overall user experience.





The outcomes demonstrate the successful development of our movie enthusiast web application platform. The backend API, the integrated MongoDB database and the user-friendly frontend interface collectively provide a functional and intuitive platform for movie enthusiasts to explore movies, post comments, and engage in discussions.

7. Advantages & Disadvantages

The key advantages of our platform include:

i. Comprehensive Movie Database: Our platform aims to provide an extensive and up-to-date movie database, allowing users to explore and discover new movies easily.

ii. User-generated Reviews: Users can post and check reviews, creating a community-driven environment that enhances the credibility and authenticity of the review system.

iii. Java Spring Boot Backend: The use of Java Spring Boot simplifies development, ensures robustness, and enables scalability for the backend API.

iv. React Frontend: React brings component reusability, efficient rendering, and a responsive user interface to the frontend web application.

v. Loosely Coupled Architecture: Our platform's loosely coupled architecture enhances flexibility and scalability, allowing for seamless integration with different components and future enhancements.

vi. MongoDB Database: MongoDB offers scalability, high availability, and flexibility for efficient storage and retrieval of movie-related data.

vii. Docker and Kubernetes Deployment: Deployment using Docker and Kubernetes provides containerization and orchestration capabilities for efficient scaling, management, and deployment across multiple environments.

Disadvantages of the platform:

i. Dependency on User Reviews: While user-generated reviews can provide valuable insights, they can also be subjective and vary in quality. Relying solely on user reviews may introduce bias or inaccurate information, potentially affecting the reliability of the review system.

ii. Limited Data Validation: As the platform allows users to post reviews, there may be a lack of strict data validation mechanisms. This could result in the presence of spam, fake, or misleading reviews, which may impact the overall credibility of the review system.

iii. Performance Considerations: While MongoDB offers scalability, its performance might be affected when handling large datasets or complex queries. Proper indexing, query optimization, and data caching strategies need to be implemented to ensure efficient and responsive data retrieval.

iv. Security Concerns: As with any web application, security is a critical consideration. The platform should address vulnerabilities such as authentication, authorization, input validation, and protection against common attacks like cross-site scripting (XSS) and SQL injection.

8. Applications

This is a versatile platform that provides valuable information and enhancing user experiences, making it suitable for various domains within the movie industry.

i. Review Platform: This solution is ideal for review platforms for all types of media including movies, video games, TV shows, music, as well as other products and services like streaming services, audio and video systems, etc., allowing users to access and share reviews, ratings, and information.
ii. Entertainment Websites and Streaming platforms: Integrating this platform into entertainment websites enhances the user experience by offering additional information, ratings, and reviews for the available movies. This can help users make informed choices.
iii. Social Media Integration: By integrating this solution with social media platforms, users can easily share movie preferences, recommendations, and reviews with their network, fostering discussions and engagement.
iv. Movie Recommendation Engines: Leveraging user reviews and ratings, this platform can power personalized movie recommendation engines, offering tailored movie suggestions to users.
v. Film Festivals and Events: This solution can be used by film festivals and events to showcase films, facilitate reviews, and provide comprehensive information about schedules and screenings.
vi. Research and Analysis: Researchers and analysts in the film industry can utilize this platform's movie database and review system to gain insights, track trends, and perform analyses on audience preferences and critical reception.

9. Conclusion

This project has successfully developed a comprehensive movies and reviews platform, employing a loosely coupled architecture with Java Spring Boot for the backend API and React for the frontend web application. The use of MongoDB as the database ensures efficient storage and retrieval of movie-related data. The deployment of the application using Docker and Kubernetes further ensures scalability and flexibility in the future.

Throughout the project, we conducted a thorough literature survey, analysing existing problems and solutions in the domain of movie review systems. By leveraging the strengths of popular platforms such as IMDb, Rotten Tomatoes, and Metacritic, we designed our platform to provide a user-friendly interface for accessing and sharing movie reviews, ratings, and information.

Our platform exhibits several technical advantages, including the robustness and scalability of Java Spring Boot, the interactive and responsive user interface of React, and the flexibility of MongoDB for handling varying data structures. The loosely coupled architecture allows for modularity and future integrations.

10.Future Scope

The future holds immense potential for enhancing the functionality and user experience of our platform. By incorporating advanced recommendation algorithms, we can further personalize movie suggestions, offering users tailored recommendations based on their preferences and behaviour.

To improve movie discovery, we plan to implement advanced search and filtering options. This will allow users to find movies based on specific criteria such as genre, cast, director, release year, or user ratings. Additionally, user profile customization will enable users to curate their own movie lists, highlight favourite genres, and receive personalized recommendations based on their preferences.

Recognizing the importance of mobile accessibility, we aim to develop mobile applications for both Android and iOS platforms. This will empower users to access the platform and receive movie recommendations on the go, ensuring a seamless and convenient experience.

Furthermore, integrating external review sources, such as professional critics' reviews or popular movie blogs, will provide a diverse range of opinions and perspectives on movies. This will enrich the platform's content and enable users to make more informed decisions. Additionally, we plan to introduce multi-language support, enabling users from different regions to access movie information and reviews in their preferred language.

By pursuing these future enhancements, we aim to solidify our platform as a comprehensive and dynamic solution for movie enthusiasts, delivering a personalized, engaging, and accessible movie experience.

11. Bibliography

"IMDb: Ratings, Reviews, and Where to Watch the Best Movies & TV Shows." *IMDb*, 2023, www.imdb.com/.

“Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews.”, *Rotten Tomatoes*, 2023, www.rottentomatoes.com/.

“Metacritic: Movie Reviews, TV Reviews, Game Reviews, and Music Reviews.” *Metacritic*, 2023, www.metacritic.com/.

Sharma, Sourabh. *Modern API Development with Spring and Spring Boot: Design highly scalable and maintainable APIs with REST, gRPC, GraphQL, and the reactive paradigm*. Packt Publishing Ltd, 2021.

Chiarelli, Andrea. *Beginning React: Simplify your frontend development workflow and enhance the user experience of your applications with React*. Packt Publishing Ltd, 2018.

Filip, Petr, and Lukáš Čegan. "Comparison of mysql and mongodb with focus on performance." *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. IEEE, 2020.

Ma, Meng, et al. "Light-weight and scalable hierarchical-MVC architecture for cloud web applications." *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2019.

Shah, Jay, and Dushyant Dubaria. "Building modern clouds: using docker, kubernetes & Google cloud platform." *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019.

Manek, Asha S., P. Deepa Shenoy, and M. Chandra Mohan. "Aspect term extraction for sentiment analysis in large movie reviews using Gini Index feature selection method and SVM classifier." *World wide web* 20 (2017): 135-154.

Banker, Kyle, et al. *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster, 2016.

Luksa, Marko. *Kubernetes in action*. Simon and Schuster, 2017.

APPENDIX

A. Source Code

BackEnd(Spring Boot)

1. Movie Controller Class-

```
package dev.movieproject.movies;
import org.bson.types.ObjectId;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/api/v1/movies")
public class MovieController {
    @Autowired
    private MovieService movieService;
    @GetMapping
    public ResponseEntity<List<Movie>> getAllMovies(){
        return new ResponseEntity<List<Movie>>(movieService.allMovies(), HttpStatus.OK);

    }
    @GetMapping ("/{imdbId}")
    public ResponseEntity<Optional<Movie>> getSingleMovie(@PathVariable String imdbId){
        return new ResponseEntity<Optional<Movie>>(movieService.singleMovie(imdbId),HttpStatus.OK);

    }
}
```

2. Movie Service Class-

```
package dev.movieproject.movies;

import org.bson.types.ObjectId;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Repository;
import org.springframework.stereotype.Service;

import javax.swing.text.html.Option;
import java.util.List;
import java.util.Optional;

@Service
public class MovieService {
    @Autowired
    private MovieReporisitory movieReporisitory;
    public List<Movie> allMovies(){
        return movieReporisitory.findAll();
    }
    public Optional<Movie> singleMovie(String imdbId){
        return movieReporisitory.findMoviesByImdbId(imdbId);

    }

}
```

3. Movie Repository Class-

```
@Repository
public interface MovieReporisitory extends MongoRepository<Movie, ObjectId> {
    Optional<Movie> findMoviesByImdbId(String imdbId);
}
```

4. Movie Review Controller Class-

```
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/api/v1/reviews")
public class ReviewController {
    @Autowired
    private RivewService rivewService;
    @PostMapping
    public ResponseEntity<Review> createReview(@RequestBody Map<String,String> payload){
        return new ResponseEntity<Review>(rivewService.createReview(payload.get("reviewBody"),payload.get("imdbId")),
HttpStatus.OK);
    }
}
```

Review Service Class-

```
@Service
public class RivewService {
    @Autowired
    private ReviewRepository reviewRepository;
    @Autowired
    private MongoTemplate mongoTemplate;
    public Review createReview(String reviewBody,String imdbId){
        Review review = reviewRepository.insert(new Review(reviewBody));
        mongoTemplate.update(Movie.class).matching(Criteria.where("imdbId").is(imdbId)).apply(new
Update().push("reviewIds").value(review)).first();
        return review;

    }
}
```

Review Class-

```
@Document(collection = "reviews")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Review {
    @Id
    private ObjectId id;
    private String body;

    public Review(String body) {
        this.body = body;
    }
}
```

Front End Class

App.js -

```
function App() {

  const [movies, setMovies] = useState();
  const [movie, setMovie] = useState();
  const [reviews, setReviews] = useState([]);

  const getMovies = async () =>{

    try
    {
      const response = await api.get("/api/v1/movies");
      setMovies(response.data);
    }
    catch(err)
    {
      console.log(err);
    }
  }
  const getMovieData = async (movieId) => {

    try
    {
      const response = await api.get(`/api/v1/movies/${movieId}`);

      const singleMovie = response.data;

      setMovie(singleMovie);

      setReviews(singleMovie.reviews);
    }
    catch (error)
    {
      console.error(error);
    }
  }
  useEffect(() => {
    getMovies();
  },[])
  return (
    <div className="App">
      <Header/>
      <Routes>
        <Route path="/" element={<Layout/>}>
          <Route path="/" element={<Home movies={movies} />} ></Route>
          <Route path="/Trailer/:ytTrailerId" element={<Trailer/>}></Route>
          <Route path="/Reviews/:movieId" element ={<Reviews getMovieData = {getMovieData} movie={movie} reviews ={reviews}
setReviews = {setReviews} />}></Route>
          <Route path="*" element = {<NotFound/>}></Route>
        </Route>
      </Routes>

    </div>
```

```
  );
}

export default App;
```

Header.js -

```
const Header = () => {

  return (
    <Navbar bg="dark" variant="dark" expand="lg">
      <Container fluid>
        <Navbar.Brand href="/" style={{"color":'gold'}}>
          <FontAwesomeIcon icon ={faVideoSlash}/>Gold
        </Navbar.Brand>
        <Navbar.Toggle aria-controls="navbarScroll" />
        <Navbar.Collapse id="navbarScroll">
          <Nav
            className="me-auto my-2 my-lg-0"
            style={{maxHeight: '100px'}}
            navbarScroll
          >
            <NavLink className ="nav-link" to="/">Home</NavLink>
            <NavLink className ="nav-link" to="/watchList">Watch List</NavLink>
          </Nav>
          <Button variant="outline-info" className="me-2">Login</Button>
          <Button variant="outline-info">Register</Button>
        </Navbar.Collapse>
      </Container>
    </Navbar>
  )
}

export default Header
```

Hero.js -

```
const Hero = ({movies}) => {

    const navigate = useNavigate();

    function reviews(movieId)
    {
        navigate(`/Reviews/${movieId}`);
    }

    return (
        <div className ='movie-carousel-container'>
            <Carousel>
                {
                    movies?.map((movie) =>{
                        return(
                            <Paper key={movie.imdbId}>
                                <div className = 'movie-card-container'>
                                    <div className="movie-card" style={{"--img": `url(${movie.backdrops[0]})`}}>
                                        <div className="movie-detail">
                                            <div className="movie-poster">
                                                <img src={movie.poster} alt="" />
                                            </div>
                                            <div className="movie-title">
                                                <h4>{movie.title}</h4>
                                            </div>
                                            <div className="movie-buttons-container">
                                                <Link to={`/Trailer/${movie.trailerLink.substring(movie.trailerLink.length - 11)}`}>
                                                    <div className="play-button-icon-container">
                                                        <FontAwesomeIcon className="play-button-icon"
                                                                icon = {faCirclePlay}
                                                        />
                                                    </div>
                                                </Link>

                                                <div className="movie-review-button-container">
                                                    <Button variant ="info" onClick={() => reviews(movie.imdbId)} >Reviews</Button>
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                </div>
                            </Paper>
                        )
                    })
                }
            </Carousel>
        </div>
    )
}
```

```
export default Hero
```