RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

FULL STACK MOVIE REVIEW WEB APPLICATION

CS23333-OBJECT ORIENTED PROGRAMMING USING JAVA

Submitted by

Vishrudha K (231501187)

Vijay R (231501183)

Of

BACHELOR OF TECHNOLOGY IN
Artificial intelligence Machine learning

BONAFIDECERTIFICATE

Certified that this project titled “Full Stack Movie Review Web- App” is the Bonafide work of **VISHRUDHA K 231501187, VIJAY R 231501183** who carried out the project work under my supervision.

Signature of Faculty – in – Charge

Submitted for the Practical Examination held on -------------------

Internal Examiner **External Examiner**

Date:26.11.2024

TABLE OF CONTENTS

1. Full Stack Movie Review Web Application

1.1 Abstract

The Movie Review Web Application is a robust platform designed to allow users to view, review, and rate movies seamlessly. Using a modern and responsive design, this application is built with HTML, CSS, Bootstrap, and Spring Boot on the backend, and it utilizes MongoDB as the NoSQL database for data storage.

The application architecture follows a client-server model where the frontend serves as the user interface, interacting with backend APIs to fetch and manage movie information and reviews. MongoDB's NoSQL capabilities make it ideal for handling varying data structures, such as storing multiple reviews associated with each movie. The app offers a clean, user-friendly interface and supports essential features like viewing movie information, reading reviews, and submitting new reviews with ratings. This application is an excellent example of how modern web technologies can come together to create a functional, interactive, and scalable solution reliable.

1.2 Introduction

 Objective: To develop a movie review web application where users can view movie information and submit reviews.

 Features:

- User-friendly interface
- Interactive modal for viewing movie details
- Secure storage of reviews in MongoDB

1.3 Purpose

The main purpose of this project is to:

- Develop an interactive web application where users can review and rate movies.
- Provide a simple interface that collects, manages, and displays user reviews efficiently.
- Demonstrate a full-stack application using HTML, CSS, Bootstrap, Java Spring Boot, and MongoDB.

1.4 Scope of the Project

This project encompasses the development of a movie review web application, including a user-friendly interface, backend API, and a MongoDB database for persistent data storage. The frontend enables users to navigate through movie listings, view detailed movie information, and add their reviews. On the backend, a series of API endpoints handle HTTP requests to ensure smooth data flow between the client and server. The MongoDB database serves as the storage system, housing details for each movie along with its associated reviews. The application is also designed to be responsive, enabling it to function seamlessly on mobile devices, desktops, and tablets. Future scalability is a key consideration, ensuring that the project can be expanded to include additional features such as personalized recommendations or user authentication.

1.5 Software Requirement Specification

1. **Database Integration:**
   - **SQL Database:** The system uses a Mongo DB database to store student data, including names and contact details. This database is connected using JDBC (Java Database Connectivity), ensuring seamless interaction between the application and the database.
   - **Schema Design:** The database schema includes main entities are Movie and Review. The Movie entity includes attributes such as Movie ID, Name, Description, and Poster URL..

2. **Backend Development:**
   - **Server Setup:** The backend server is implemented using Java with the HttpServer class, handling various endpoints for different functionalities.
   - **Endpoints:**
     - /api/movies: Handles to display all movies.
     - /api/reviews: Inserting the review for a particular movie with object id

3. **Frontend Development:**
    - **HTML and CSS:** The frontend is developed using HTML for structure and CSS for styling, providing a user-friendly interface for managing student information.
    - **JavaScript:** Utilizes JavaScript to dynamically update the frontend, handle form submissions, and ensure interactive user experiences.
4. **Functionality:**

The frontend leverages HTML5, CSS3, and Bootstrap to create an attractive, responsive layout that is easy to navigate. On the backend, Java with Spring Boot serves as the foundation for building and managing RESTful endpoints. MongoDB is utilized as the database, chosen for its flexibility and efficiency in handling document-based data structures. Development tools such as Eclipse or IntelliJ were employed to manage code, while Postman assisted in testing the API endpoints. The embedded Tomcat server within Spring Boot simplifies deployment and serves as the local server for hosting the application.

Technology Stack:

- **Frontend:** HTML, CSS, JavaScript
- **Backend:** Java, SQL, JDBC
- **Tools and Libraries:** JSON handling libraries for data exchange, HttpServer for handling HTTP requests and responses.

Overall Description

Product Perspective

The system is designed using a client/server architecture, ensuring compatibility across various operating systems. The frontend is developed with HTML, CSS, and JavaScript, utilizing libraries like Bootstrap for responsive design and Chart.js for data visualization. The backend is powered by Java with Spring Boot for handling HTTP requests, and MongoDB for data storage.

Product Functionality

a) View Movie List: Allows users to access and browse a comprehensive list of movies available in the database.

b) View Movie Details: Enables users to click on a specific movie to view detailed information, including descriptions, ratings, and reviews.

c) Submit Reviews: Provides the capability for users to submit reviews for movies, including their names, ratings, and comments.

d) Search Functionality: Allows users to search for movies based on various criteria such as title, genre, or release year.

e) User Authentication (Optional): While the system allows anyone to submit reviews, user authentication can be implemented to manage user access and enhance security.

User and Characteristics

Qualification:

- Users should have a basic understanding of movie genres and a general interest in film.

Experience:

- Familiarity with online platforms and basic data entry is beneficial.

Technical Skills:

Users are expected to have elementary knowledge of computers and the ability to interact with web-based applications.

Operating Environment:

Hardware Requirements:

- **Processor**: Any processor over i3
- **Operating System**: Windows 8, 10, 11
- **Processor Speed**: 2.0 GHz
- **RAM**: 4GB
- **Hard Disk**: 500GB

Software Requirements:

- **Database**: MongoDB
- **Frontend**: HTML, CSS, JavaScript (with app.js)
- **Backend**: Java(Spring boot)

Constraints

- **Authorized Access:** While anyone can submit reviews, certain administrative functions may be limited to authorized users, such as administrators.
- **Delete Operation:** The delete operation for reviews may be restricted to administrators to prevent accidental data loss.
- **Data Consistency:** Administrators must exercise caution during deletion to ensure data consistency and prevent unintended data loss.

Assumptions and Dependencies

- System administrators are responsible for creating and securely communicating login credentials to users if authentication is implemented.
- Users are assumed to have basic knowledge of using web-based applications.

Specific Requirements

User Interface Features for the User Management System of Students Are:

- **View Movie List:** Display a comprehensive list of movies available for review.
- **View Movie Details:** Show detailed information about selected movies, including reviews and ratings.
- **Submit Reviews:** Allow users to input and store their reviews accurately and efficiently.
- **Search Movies:** Retrieve specific movie details quickly based on search criteria.
- **User Authentication (Optional):** Ensure secure management of user access to protect sensitive information.

Hardware Interface:

- Screen resolution of at least 640 x 480 or above.
- Compatible with any version of Windows 8, 10,11

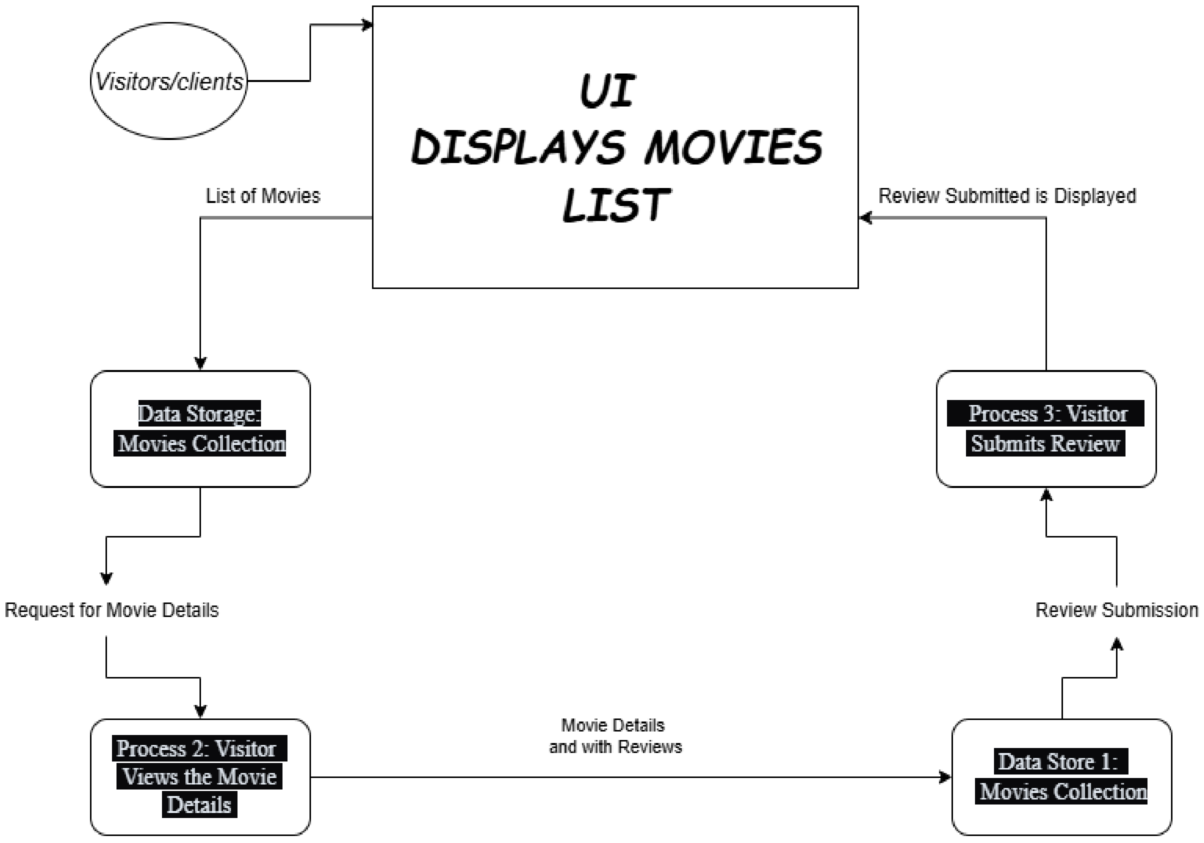
Software Interface for Student Mark Analysis System

- **Operating System**: MS-Windows (Windows 8, 10, 11)
- **Frontend Development**: HTML, CSS, JavaScript (with Chart.js)
- **Backend Development**: Java
- **Database**: SQL

2. SYSTEM FLOW DIAGRAMS

2.1 Use Case

Diagrams :



Visitors/clients
UI DISPLAYS MOVIES LIST
List of Movies
Review Submitted is Displayed
Data Storage: Movies Collection
Process 3: Visitor Submits Review
Request for Movie Details
Review Submission
Process 2: Visitor Views the Movie Details
Movie Details and with Reviews
Data Store 1: Movies Collection

FIG 2.1 CASE DIAGRAM

MODULE DESCRIPTION

The application is divided into three primary modules. The User Module provides a seamless experience for users to browse movies and submit reviews. The Movie Module manages the listing and display of movie details, including descriptions, ratings, and posters. Finally, the Review Module collects and stores user reviews in the database. This modular architecture allows each component to operate independently, facilitating ease of maintenance and scalability.

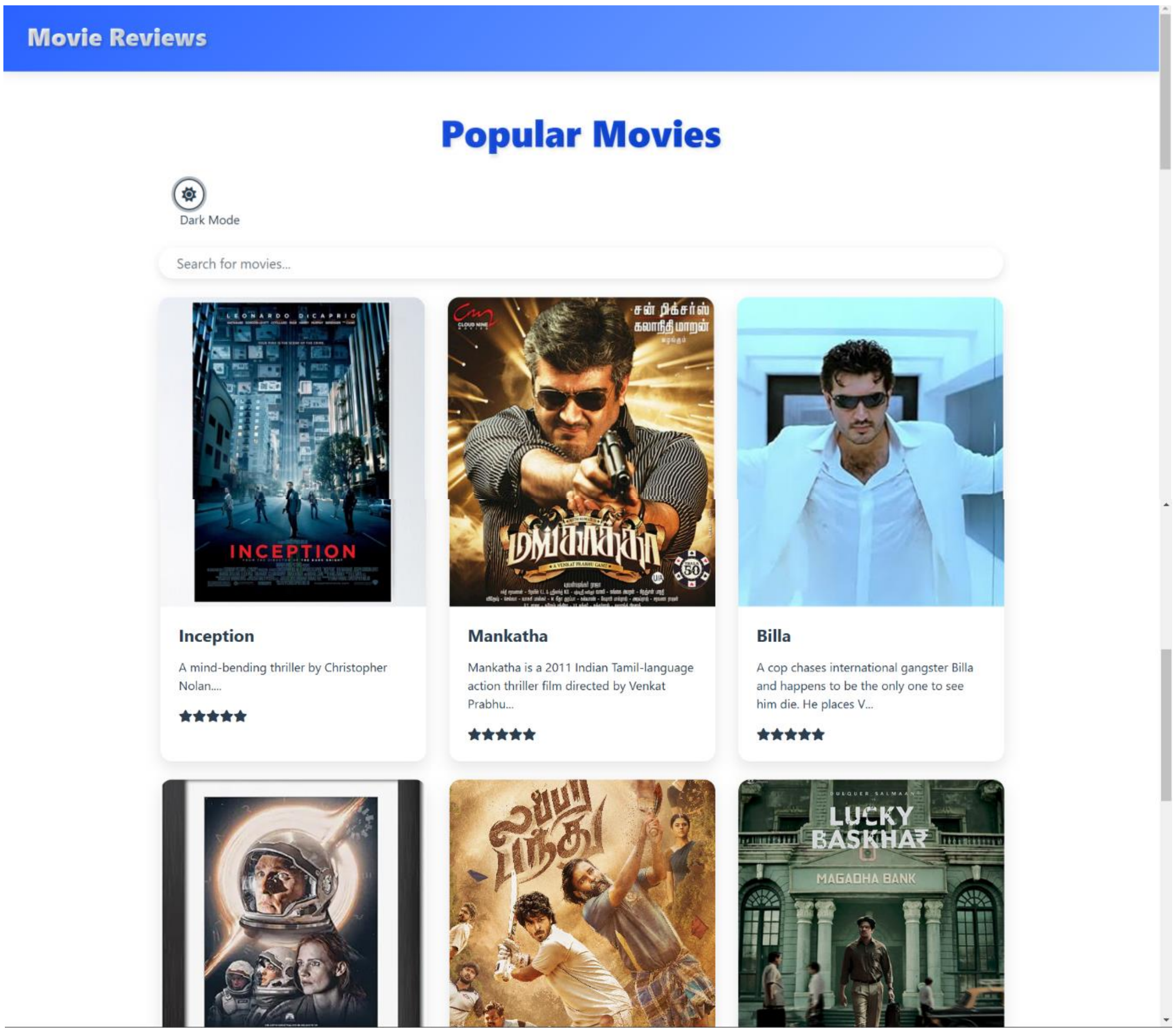
3.IMPLEMENTATION

3.1 Design:

The application follows a multi-tiered architecture, separating the frontend, backend, and database layers to enhance maintainability and scalability.

- **Frontend**: HTML, CSS, and Bootstrap for styling and responsiveness.
- **Backend**: Spring Boot API with RESTful endpoints handling all data requests.
- **Database**: MongoDB for document-based storage of movie details and reviews.

3.2 Movies Collection User Interface:



Inception

A mind-bending thriller by Christopher Nolan....

★★★★★

Mankatha

Mankatha is a 2011 Indian Tamil-language action thriller film directed by Venkat Prabhu...

★★★★★

Billa

A cop chases international gangster Billa and happens to be the only one to see him die. He places V...

★★★★★

Figure 3.1 Client UI

3.3 Movie Modal Code

```java
package com.example.moviereview.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.List;

@Document(collection = "movies")
public class Movie {
    @Id
    private String id;
    private String name;
    private String description;
    private String posterUrl;
    private List<Review> reviews;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPosterUrl() {
        return posterUrl;
    }

    public void setPosterUrl(String posterUrl) {
        this.posterUrl = posterUrl;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public List<Review> getReviews() {
        return reviews;
    }

    public void setReviews(List<Review> reviews) {
        this.reviews = reviews;
    }
// Constructors, getters, setters
}
```

3.4 API Controller(Controlees the End Points) :

```java
package com.example.moviereview.controller;

import com.example.moviereview.model.Movie;
import com.example.moviereview.model.Review;
import com.example.moviereview.service.MovieService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/movies")
public class MovieController {
    @Autowired
    private MovieService movieService;

    @GetMapping
    public List<Movie> getAllMovies() {
        return movieService.getAllMovies();
    }

    @GetMapping("/{id}")
    public Movie getMovieById(@PathVariable String id) {
        return movieService.getMovieById(id);
    }

    @PostMapping
    public Movie addMovie(@RequestBody Movie movie) {
        return movieService.addMovie(movie);
    }

    @PostMapping("/{id}/review")
    public Movie addReview(@PathVariable String id, @RequestBody Review review) {
        return movieService.addReview(id, review);
    }
}
```

3.5 Service Class:

```java
package com.example.moviereview.service;

import com.example.moviereview.model.Movie;
import com.example.moviereview.model.Review;
import com.example.moviereview.repository.MovieRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class MovieService {
    @Autowired
    private MovieRepository movieRepository;

    public List<Movie> getAllMovies() {
        return movieRepository.findAll();
    }

    public Movie getMovieById(String id) {
        return movieRepository.findById(id).orElse(null);
    }

    public Movie addMovie(Movie movie) {
        return movieRepository.save(movie);
    }

    public Movie addReview(String movieId, Review review) {
        Movie movie = getMovieById(movieId);
        if (movie != null) {
            movie.getReviews().add(review);
            movieRepository.save(movie);
        }
        return movie;
    }
}
```

3.6 Movie Review HTML Review Form:

```html
<form id="reviewForm">
  <input type="text" name="reviewerName" placeholder="Your Name" required>
  <input type="number" name="rating" min="0" max="5" step="0.5" required>
  <textarea name="content" placeholder="Write your review" required></textarea>
  <button type="submit">Submit</button>
</form>
```

3.7 Styling with CSS:

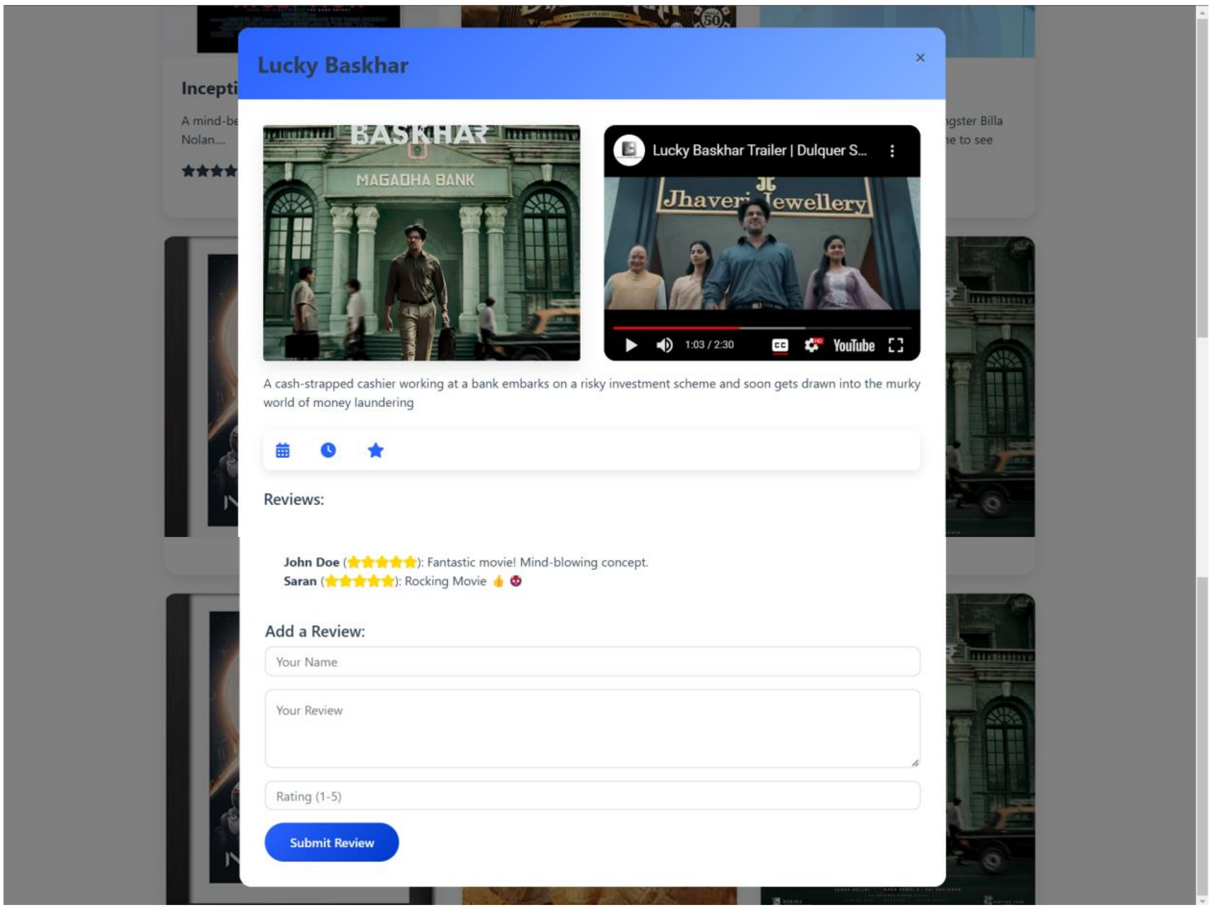
```css
/* Card Styling */
#movie-list .card {
    background-color: var(--card-bg);
    border: none;
    border-radius: 15px;
    overflow: hidden;
    box-shadow: 0 8px 20px var(--shadow-color);
    transition: all 0.4s cubic-bezier(0.175, 0.885, 0.32, 1.275);
}

#movie-list .card:hover {
    transform: var(--card-hover-transform);
    box-shadow: 0 15px 30px var(--shadow-color);
}

.card-img-top {
    height: 400px;
    object-fit: cover;
    transition: transform 0.4s ease;
}

.card:hover .card-img-top {
    transform: scale(1.05);
}
```

3.8 Review Page:



Lucky Baskhar

×

Lucky Baskhar Trailer | Dulquer S...

A cash-strapped cashier working at a bank embarks on a risky investment scheme and soon gets drawn into the murky world of money laundering

Reviews:

John Doe (⭐⭐⭐⭐⭐): Fantastic movie! Mind-blowing concept.
Saran (⭐⭐⭐⭐⭐): Rocking Movie 👍 😈

Add a Review:

Your Name

Your Review

Rating (1-5)

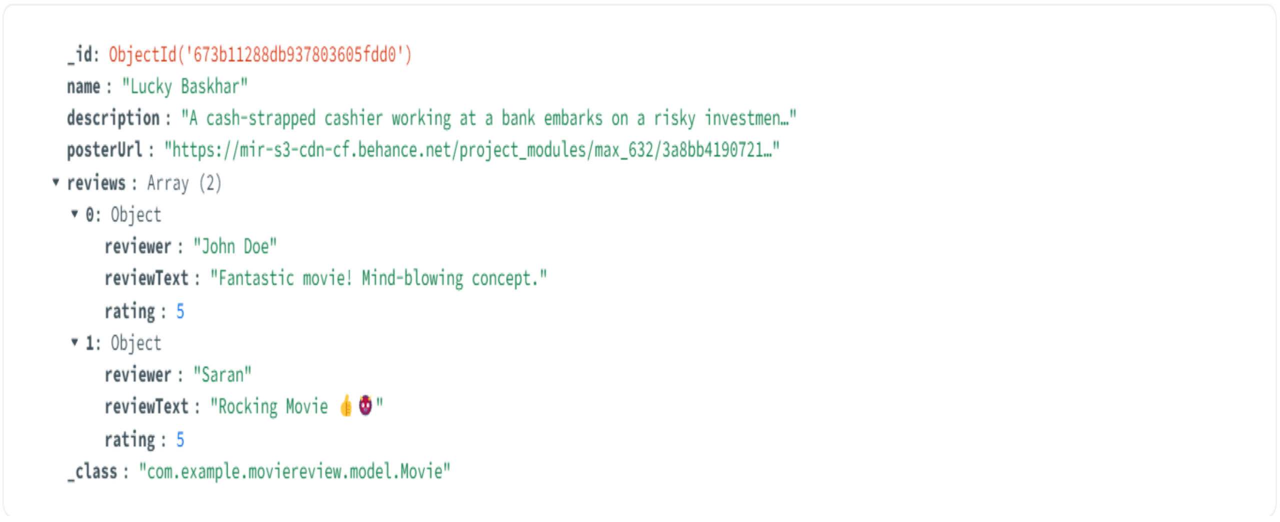
Submit Review

Figure 3.2 Review Page

3.9 Database Design :

MongoDB stores data in two collections:

1. **Movies Collection**: Holds documents for each movie, including details and associated reviews.
2. **Reviews Collection**: Embedded within each movie document to store user reviews.



```
_id: ObjectId('673b11288db937803605fdd0')
name : "Lucky Baskhar"
description : "A cash-strapped cashier working at a bank embarks on a risky investmen…"
posterUrl : "https://mir-s3-cdn-cf.behance.net/project_modules/max_632/3a8bb4190721…"
▾ reviews : Array (2)
  ▾ 0: Object
      reviewer : "John Doe"
      reviewText : "Fantastic movie! Mind-blowing concept."
      rating : 5
  ▾ 1: Object
      reviewer : "Saran"
      reviewText : "Rocking Movie 👍👹"
      rating : 5
_class : "com.example.moviereview.model.Movie"
```

Figure 3.3Mongo DB Collection Document

Sample Database Schema Structure:

```
{
  "movieId": "unique_movie_id",
  "name": "Inception",
  "description": "A mind-bending thriller...",
  "posterUrl": "url_to_poster_image",
  "reviews": [
     {
        "reviewerName": "John Doe",
        "rating": 4.5,
        "content": "An amazing experience."
     }
  ]
}
```

4.Features

4.1 User Features:

1. **Search Functionality:**
   - **Description:** Users can search for movies using various criteria, primarily by title. The search functionality is designed to be intuitive and user-friendly, allowing users to quickly find the movies they are interested in.
   - **Implementation Details:**
     - **Search Bar:** A prominently placed search bar on the homepage allows users to enter keywords related to the movie title.
     - **Autocomplete Suggestions:** As users type, the system provides autocomplete suggestions based on existing movie titles in the database, enhancing the search experience.
     - **Filters:** Users can refine their search results using filters such as genre, release year, and rating to narrow down their options effectively.
     - **Search Results Display:** The results are displayed in a grid or list format, showing movie posters, titles, and brief descriptions, making it easy for users to browse through the options.

2. **Movie Details:**

Description: When users click on a specific movie from the search results or the movie list, they are taken to a dedicated movie details page that provides comprehensive information about the selected film.

Implementation Details:

Detailed Information: The movie details page includes essential information such as the movie's title, genre, release date, director, cast, and a synopsis.

Trailer: An embedded video player allows users to watch the official movie trailer directly on the page, providing a visual preview of the film.

User Reviews: A section dedicated to user reviews displays ratings and comments submitted by other users, allowing potential viewers to gauge the movie's reception.

Rating System: Users can see the average rating based on all submitted reviews, giving them a quick overview of the movie's popularity.

1. **Review Submission:**
    - **Description:** Users have the ability to submit their own reviews and ratings for movies they have watched, contributing to the community and sharing their opinions.
    - **Implementation Details:**
        - **Review Form:** A user-friendly form allows users to enter their review text and select a star rating (e.g., 1 to 5 stars) for the movie.
        - **Name Display:** Users can choose to display their name alongside their review, fostering a sense of community and accountability.
        - **Submission Confirmation:** After submitting a review, users receive a confirmation message indicating that their review has been successfully submitted and will be visible after moderation (if applicable).
        - **Review Editing:** Users can edit or delete their reviews within a specified time frame, allowing them to update their opinions as needed.

4.2 Admin Features

1. **Movie Management:**
    - **Description:** Admins have the authority to manage the movie database, ensuring that the information is accurate, up-to-date, and relevant.
    - **Implementation Details:**
        - **Add Movie:** Admins can add new movie entries by filling out a form that includes fields for the title, genre, release date, synopsis, cast, and trailer URL.
        - **Update Movie:** Admins can edit existing movie entries to correct information or update details as necessary. This includes changing the synopsis, updating the cast list, or modifying the trailer link.
        - **Delete Movie:** Admins can remove movies from the database that are no longer relevant or have been incorrectly added. A confirmation prompt ensures that deletions are intentional to prevent accidental loss of data.

    - **Bulk Upload:** Admins can also have the option to bulk upload movie data using CSV or Excel files, streamlining the process of adding multiple entries at once.

2. **Review Moderation:**

- **Description:** Admins are responsible for overseeing user-submitted reviews to maintain the quality and appropriateness of content on the platform.

- **Implementation Details:**

    - **Review Dashboard:** Admins have access to a dedicated dashboard that lists all user-submitted reviews, categorized by movie, with options to filter by status (pending, approved, flagged).
    - **Approval Process:** Admins can review each submission and decide whether to approve it for public viewing, ensuring that all content adheres to community guidelines and standards.
    - **Flagging System:** Users can flag inappropriate reviews, which will be brought to the attention of admins for further review. Admins can then take appropriate action, such as editing or deleting flagged reviews.
    - **Analytics:** Admins can view analytics related to reviews, such as the number of reviews submitted per movie, average ratings, and trends over time, helping them understand user engagement and movie popularity.

5. Testing

Testing is a crucial phase in the software development lifecycle that ensures the application functions correctly and meets user requirements. For the movie review project, a variety of testing methods will be employed.

5.1 Types of Testing

1. **Unit Testing:**
   - Tests individual components or functions in isolation.
   - Framework: JUnit for backend functions (e.g., movie retrieval, review submission).
2. **Integration Testing:**
   - Verifies interactions between different modules.
   - Ensures frontend and backend communicate correctly (e.g., submitting reviews).
3. **Functional Testing:**
   - Evaluates the application against functional requirements.
   - Test cases for user features like searching for movies and submitting reviews.
4. **User Acceptance Testing (UAT):**
   - Validates the application from the end-user's perspective.
   - Involves real users testing the application and providing feedback.
5. **Performance Testing:**
   - Assesses responsiveness and stability under load.
   - Tools like Apache JMeter will simulate multiple users accessing the application.
6. **Security Testing:**
   - Identifies vulnerabilities in the application.

- Includes penetration testing to ensure data protection and secure user authentication.

7. **Regression Testing:**
    - Ensures new code changes do not affect existing functionality.
    - Automated tests will be run after updates to verify system stability.

5.2 Specific Test Cases

1. **User Features:**
    - **Search Functionality:** Verify that users can search for movies by title and receive accurate results.
    - **Movie Details:** Ensure clicking on a movie displays the correct details, including the trailer and user reviews.
2. **Review Submission:**
    - Confirm that users can submit reviews and that they appear on the movie details page after moderation.
3. **Admin Features:**
    - **Movie Management:** Test that admins can add, update, and delete movie entries successfully.
    - **Review Moderation:** Verify that admins can approve or reject user-submitted reviews.
4. **Performance:**
    - Test the application’s ability to handle multiple concurrent users without significant delays.
5. **Security:**
    - Ensure that user input is properly validated to prevent SQL injection and other vulnerabilities.

By implementing these testing methods and strategies, the movie review project will ensure a reliable, user-friendly, and secure application.

6. Conclusion

Summary

Diagram Description User Interaction Layer:
The user accesses the application through a web interface, interacting with HTML/CSS pages enhanced by Bootstrap. Key actions include viewing the movie list, viewing detailed movie information, and submitting reviews.

API Layer (Backend):
The frontend communicates with the backend via RESTful APIs built with Spring Boot. Backend endpoints process HTTP requests for retrieving movies, retrieving movie details, and posting reviews.
Data Layer (Database):
MongoDB stores data in two main collections: Movies: Contains fields for movie details like name, description, poster URL, and reviews. Reviews: Each review includes a rating, reviewer's name, and review content. Reviews are nested within each movie document in the movies collection. The backend retrieves movie data from MongoDB and returns it to the frontend, enabling dynamic page rendering. Sample Data Flow Diagram (Textual Explanation)
Process 1: User Requests Movie List
Input: User accesses the movie list page. Data Flow: Request is sent from the user to the backend API (GET /api/movies). The backend retrieves the list of movies from MongoDB. The list of movies is returned to the frontend and displayed to the user**.**
Process 2: User Views Movie Details
Input: User clicks on a specific movie to view details. Data Flow: The frontend sends a GET request to /api/movies/{id}. The backend retrieves movie details and reviews from MongoDB. The frontend renders the movie details, including the poster, description, and reviews.
Process 3: User Submits Review
Input: User fills out the review form and submits it. Data Flow: The frontend sends a POST request with review details to /api/movies/{id}/review. The backend adds the review to the movie's review array in MongoDB. MongoDB

updates the movie document with the new review, and the backend confirms submission to the frontend.

7.Refrences

 Oracle Corporation. (2023). *Java Platform, Enterprise Edition (Java EE).* Oracle Corporation. Retrieved from https://www.oracle.com/java/technologies/
This resource offers comprehensive documentation on Java EE, which provides insight into building scalable server-side applications and managing HTTP requests effectively.

 MongoDB, Inc. (2023). *MongoDB Documentation.* MongoDB, Inc. Retrieved from https://www.mongodb.com/docs/
An essential resource for understanding document-based databases, MongoDB's official documentation includes guides on database operations, security, and best practices for building scalable applications.

 Bootstrap Documentation. (2023). *Bootstrap 5: CSS Framework for Responsive Web Design.* Bootstrap. Retrieved from https://getbootstrap.com/docs/5.0/
Bootstrap's documentation provides in-depth resources on utilizing responsive CSS frameworks, a critical component for designing user interfaces that adapt to various devices in a modern application.

 W3Schools. (2023). *HTML, CSS, and JavaScript Tutorials.* W3Schools. Retrieved from https://www.w3schools.com/
W3Schools offers foundational tutorials on HTML, CSS, and JavaScript, supporting developers in implementing frontend components and interactions in a structured manner for web applications.

 Spring Boot Documentation. (2023). *Spring Boot Framework Guide.* Pivotal Software, Inc. Retrieved from https://spring.io/projects/spring-boot
This official documentation offers guidance on Spring Boot, which is instrumental for developing Java-based backend services with REST APIs. It includes information on project setup, database integration, and API management.

 Stack Overflow Community. (2023). *Java, MongoDB, and Web Development Q&A.* Stack Overflow. Retrieved from https://stackoverflow.com/
Stack Overflow provides community-driven solutions to programming challenges encountered during project development, especially in integrating Java backend with MongoDB and frontend technologies like HTML and CSS.

 Google Developers. (2023). *Design Principles for User Experience.* Google LLC. Retrieved from https://developers.google.com/
This resource provides user experience design principles that help developers create intuitive and visually appealing interfaces, which are essential for engaging users on a movie review platform.

 Moorhead, M. (2023). *Full-Stack Development with Java and MongoDB.* Packt Publishing.
This book provides insights into the integration of Java and MongoDB for building full-stack applications, offering real-world case studies and examples relevant to the project.

 Nielsen, J., & Norman, D. (2023). *Usability Principles for Interface Design.* Nielsen Norman Group. Retrieved from https://www.nngroup.com/articles/usability-principles/
This article highlights usability and user experience principles that can be applied to enhance the user-

friendliness of the movie review application, ensuring an accessible and satisfying experience for users.