TECHNICAL DOCUMENTATION

ON

"EduConsultancy"

Project Overview:

EduConsultancy is a comprehensive web-based platform designed to streamline educational consultancy services. It provides tools for managing courses, feedback, products, blogs, user subscriptions, and much more. This platform aims to improve the experience for both users and administrators by offering advanced features for course management, communication, and customer service.

Technologies Used:

Frontend: React, Tailwind CSS

Backend: Spring Boot

Database: MySQL

Payment Integration: Razorpay

Contents

Abstract

The increasing demand for efficient and user-friendly educational consultancy services has led to the development of the **EduConsultancy** platform. This project presents a comprehensive online system designed to streamline the process of connecting users with educational consultants, leveraging modern web technologies for enhanced functionality and scalability. Built using a full-stack approach with React for the frontend and Spring Boot for the backend, the platform offers features such as secure authentication, subscription management, and a robust admin dashboard for managing user data, feedback, blogs, and products.

The system integrates Razorpay for secure payment processing and email notifications for real-time updates on subscriptions and password resets. Future enhancements include AI-based consultancy matching to personalize recommendations, an analytics dashboard for administrators to monitor key metrics, and mobile app integration to improve accessibility. Applicable to a wide range of users, from students seeking guidance to administrators managing operations, **EduConsultancy** provides a reliable, scalable, and innovative solution to address the challenges in the education consultancy domain. This paper outlines the architecture, implementation, and potential benefits of the proposed platform, showcasing its effectiveness in transforming consultancy services.

Keywords-*EduConsultancy, education, consultancy, subscription, React, Spring Boot, Razorpay, scalability.*

Chapter 1

INTRODUCTION

1.1 Introduction

The Edu Consultancy platform is an integrated system that facilitates seamless communication between students and educational consultants, providing personalized academic and career guidance. The platform aims to make the process of consultancy more accessible and efficient, connecting students with experts who can help them navigate their educational journey. By combining modern technologies with intuitive interfaces, Edu Consultancy delivers a dynamic and scalable solution designed to cater to the evolving needs of both students and consultants.

Its purpose is to bridge the gap between students seeking educational guidance and experts in various fields, whether for higher education, career planning, or subject-specific advice. The platform enables students to explore available services, interact with consultants, and make informed decisions about their academic futures. Additionally, Edu Consultancy simplifies the administrative aspects for consultants, allowing them to manage their offerings, interact with students, and ensure a smooth user experience.

1.2 Target Audience

- **Students:** Individuals looking for educational resources and feedback.
- **Educational Institutions:** Providers of courses, feedback, and consultancy services.
- **General Users:** Anyone interested in subscribing to educational products and interacting with the platform.

1.3 Key Features

The Edu Consultancy platform offers the following key features:

- **Admin Panel Functionality:**
  - **CRUD Operations:** Admin can Create, Read, Update, and Delete:

    - Feedbacks
    - Contacts
    - Products
    - Blogs
  - **User Management:** Admin can view all users and their orders.

- **User Features:**
  - **View & Subscribe:**
    - Users can view all feedbacks, contacts, products, and blogs.
    - Users can subscribe to specific courses/products by making payments via Razorpay.
  - **Dashboard Overview:** A personalized dashboard to track:
    - Orders made by the user
    - Feedbacks posted by the user
    - Contacts created by the user
    - Overview of all blogs, products, orders, feedbacks, and contacts
  - **Content Management:** Admins can edit and delete their blogs, products, contacts, and feedback.

- **Security & Authentication:**
  - **JWT-based Authentication:** Secure login and access for users and admins.
  - **Password Reset:** OTP-based password reset functionality using email service.

- **Notifications:**
  - **Toasts for Notifications:** Real-time notifications for actions like successful operations or errors.
  - **Email Notifications:**
    - OTP for password reset.
    - Confirmation email when a user purchases a course/product.
    - Real-time Email Notifications when a user subscribes to a particular course/product, confirming the purchase and subscription details.

- **Form Validations:** Ensures all user inputs are validated for accuracy and security.

Chapter 2

System Architecture

2.1 Overview

The Edu Consultancy platform operates with a clear separation of concerns between the frontend, backend, and database. Below is an overview of how these components interact:

2.1.1 Frontend (React)

- The frontend is built using React, which is responsible for presenting the user interface and capturing user inputs.
- React communicates with the backend through **RESTful API calls** using libraries like Axios or Fetch.
- Data from the backend (such as user information, products, courses, etc.) is dynamically rendered in the frontend to provide a seamless user experience.

2.1.2 Backend (Spring Boot)

- The backend is built using **Spring Boot**, which provides a robust server environment for managing business logic and processing API requests.
- Spring Boot exposes **RESTful API endpoints** that the React frontend consumes.
- The backend processes requests such as user authentication, CRUD operations, payment handling (via Razorpay), and more.

2.1.3 Database (Relational Database - MySQL)

- The backend interacts with a **relational database** to store data such as user information, feedback, products, and order details.
- Spring Boot communicates with the database via **JPA (Java Persistence API)** and **Hibernate**, ensuring smooth data handling and query execution.
- The database is updated or queried whenever the frontend requests or submits data via API calls.

2.2 Data Flow

- The **Frontend** sends HTTP requests to the **Spring Boot backend** via API endpoints.
- The **Spring Boot backend** processes the request and communicates with the **database** to retrieve or update data.
- The **backend** sends a response back to the **frontend**, which is rendered for the user to view or interact with.

Chapter 3

Frontend Overview

The frontend of the Edu Consultancy platform is built using **React**, a popular JavaScript library for building user interfaces. React allows for the creation of dynamic, reusable components that render efficiently based on state and data changes. Below is an overview of the key technologies and major components of the frontend:

3.1 Technologies Used

- **React:** The core technology used for building the user interface. React enables the development of single-page applications with fast, dynamic updates.
- **Tailwind CSS:** A utility-first CSS framework that is used to style the frontend. Tailwind provides a flexible and efficient way to design components without having to write custom CSS, making the styling process faster and more maintainable.
- **Axios/Fetch:** These libraries are used to make API calls from the frontend to the backend. Axios is used for handling asynchronous HTTP requests, allowing React components to retrieve and send data to the Spring Boot backend.

3.2 Major Components

3.2.1 Login

The login page allows users to authenticate themselves by entering their credentials (username/email and password). It communicates with the backend via the login API endpoint, which verifies the credentials and returns a JWT (JSON Web Token) to the frontend for secure session management.

3.2.2 Registration

The registration page enables new users to create an account by providing their details such as name, email, and password. The data is sent to the backend via the registration API, and once the user is successfully created, a confirmation email may be sent to the user.

3.2.3 Password Reset

The password reset component allows users to reset their passwords via an OTP (One-Time Password) sent to their registered email. The frontend handles the UI for entering the email, verifying the OTP, and allowing the user to set a new password.

3.2.4 Dashboard

The user dashboard provides an overview of the user's activity, such as orders made, feedback posted, and subscriptions. It retrieves and displays user-specific data by calling the appropriate backend API endpoints.

3.2.5 Product and Blog Listings

These pages display the available products, courses, and blogs that users can view and interact with. They provide details about the products or blogs and allow users to subscribe or interact with them.

3.2.6 CRUD Operations for Blogs, Products, Contacts, and Feedback

Users can edit and delete their blogs, products, contacts, and feedbacks through the respective components. The frontend sends the necessary data to the backend API to perform the required CRUD operations.

3.3 Integration with Third-Party Services

3.3.1 Razorpay

The frontend integrates with **Razorpay** for handling payments when users subscribe to courses or products. Razorpay provides a secure payment gateway, and the frontend handles the user input for payment details, communicates with Razorpay's API, and updates the user's subscription status.

Chapter 4

Backend Overview

The backend of the Edu Consultancy platform is built using **Spring Boot**, a robust and widely used Java framework for creating production-grade applications. Spring Boot simplifies the setup and configuration of Spring applications by providing defaults and production-ready features. It is designed to work seamlessly with modern web technologies and offers built-in tools for handling common tasks like security, database access, and web services.

4.1 Major Components

4.1.1 Controllers

In the Spring Boot framework, controllers are responsible for handling HTTP requests and routing them to the appropriate services. They act as the interface between the client (frontend) and the server-side logic. Controllers are annotated with `@RestController` or `@Controller`, and methods inside the controller are mapped to specific HTTP endpoints using annotations like `@GetMapping`, `@PostMapping`, `@PutMapping`, etc.

For example, a login controller will handle user authentication requests, while a product controller may handle CRUD operations for products.

4.1.2 Services

Services in Spring Boot handle the business logic of the application. They act as an intermediary between controllers and repositories, ensuring that the correct operations are performed and any necessary logic is applied. Services are typically annotated with `@Service`, and they contain methods for processing data, making decisions, and interacting with the database or external systems.

For example, a `UserService` might handle logic for user registration, while an `OrderService` could manage order processing.

4.1.3 Repositories

Repositories in Spring Boot are responsible for interacting with the database. They provide methods to query, save, update, and delete data from the database. Spring

Boot uses **Spring Data JPA** (Java Persistence API) for database interaction. Repositories are typically interfaces that extend `JpaRepository` or `CrudRepository`, which come with built-in methods for database operations.

For example, a `UserRepository` may provide methods like `findByEmail`, `save`, or `deleteById` for interacting with the users' data.

4.2 Security Features

4.2.1 Authentication

Spring Boot provides built-in support for **JWT-based authentication** (JSON Web Token), which is used for secure login and API access. The authentication process typically involves validating a user's credentials (username/email and password), generating a JWT token, and sending it back to the client. The client can then use this token to authenticate subsequent API requests. Spring Security is used to configure the authentication flow and ensure that only authenticated users can access specific endpoints.

4.2.2 Password Encryption

Spring Boot uses **BCrypt** or other hashing algorithms to securely encrypt passwords before they are stored in the database. When a user registers or updates their password, the password is encrypted using the `BCryptPasswordEncoder` class before being saved in the database. During login, the entered password is hashed and compared with the stored hash to verify the user's identity.

4.3 Email Service Implementation for Password Reset

4.3.1 Password Reset

The password reset functionality in Spring Boot uses an email service to send a One-Time Password (OTP) to the user's registered email address. The email service is typically implemented using `JavaMailSender` or a third-party service like. Spring Boot provides integration with these services to send emails directly from the backend.

The process typically involves generating a unique OTP, sending it to the user's email, and allowing them to enter the OTP on the frontend to verify their identity. After verification, the user can reset their password. The email service is configured with SMTP settings or a third-party email API and can be easily integrated into the Spring Boot backend.

Chapter 5

Database design

5.1 Users Table

Stores information about users who register on the platform, including their personal details and authentication data.

- **Primary Key:** user_id
- **Attributes:**
  - `user_id`: Unique identifier for each user.
  - `created_at`: Timestamp when the user account was created.
  - `email`: User's email address (used for authentication).
  - `name`: Full name of the user.
  - `password`: Encrypted password used for login.
  - `profile_picture`: Link to the user's profile picture.
  - `role`: The role of the user (e.g., Admin, User).
  - `updated_at`: Timestamp when the user's information was last updated.
  - `username`: Unique username for the user.

5.2 Blogs Table

Stores information related to blogs published by users, including the title, content, and images.

- **Primary Key:** blog_id
- **Foreign Key:** user_id (linked to the `users` table)
- **Attributes:**
  - `blog_id`: Unique identifier for each blog post.
  - `blog_image`: Image associated with the blog post.
  - `category`: Category under which the blog falls.

  - `content`: The actual content of the blog post.
  - `created_at`: Timestamp when the blog was created.
  - `title`: Title of the blog post.
  - `updated_at`: Timestamp when the blog was last updated.
  - `user_id`: Foreign key referring to the author of the blog (linked to the `users` table).

5.3 Contacts Table

Stores contact messages submitted by users via the contact form.

- **Primary Key:** contact_id
- **Foreign Key:** user_id (linked to the `users` table)
- **Attributes:**
  - `contact_id`: Unique identifier for each contact message.
  - `created_at`: Timestamp when the contact message was submitted.
  - `email`: User's email address for communication.
  - `message`: Content of the contact message.
  - `name`: Name of the user who submitted the contact form.
  - `subject`: Subject of the contact message.
  - `updated_at`: Timestamp when the contact message was last updated.
  - `user_id`: Foreign key referring to the user who submitted the message (linked to the `users` table).

5.4 Feedbacks Table

Stores feedback submitted by users.

- **Primary Key:** feedback_id
- **Foreign Key:** user_id (linked to the `users` table)
- **Attributes:**
  - `feedback_id`: Unique identifier for each feedback.
  - `created_at`: Timestamp when the feedback was submitted.
  - `email`: User's email address who submitted the feedback.
  - `message`: Content of the feedback.
  - `name`: Name of the user who submitted the feedback.
  - `phone`: User's phone number.
  - `updated_at`: Timestamp when the feedback was last updated.
  - `user_id`: Foreign key referring to the user who submitted the feedback (linked to the `users` table).

5.5 Forgot Password Table

Stores OTPs generated for password reset functionality, with expiration times.

- **Primary Key:** fpid
- **Foreign Key:** user_user_id (linked to the `users` table)
- **Attributes:**
  - `fpid`: Unique identifier for each password reset entry.
  - `expiration_time`: Expiration time for the OTP.
  - `otp`: The one-time password (OTP) used for password reset.
  - `user_user_id`: Foreign key referring to the user requesting the password reset (linked to the `users` table).

5.6 Products Table

Stores information about products or courses available on the platform.

- **Primary Key:** product_id
- **Attributes:**
  - `product_id`: Unique identifier for each product or course.
  - `category`: Category under which the product or course falls.
  - `description`: Detailed description of the product or course.
  - `price`: Price of the product or course.
  - `product_image`: Image associated with the product or course.
  - `rating`: Rating of the product or course.
  - `title`: Title of the product or course.

5.7 Product Buyers Table

Stores information about which users have purchased which products.

- **Composite Primary Key:** product_id, user_identifier (linked to the `products` and `users` tables)
- **Attributes:**
  - `product_id`: Foreign key referring to the product purchased (linked to the `products` table).
  - `user_identifier`: Unique identifier for the user who purchased the product (linked to the `users` table).

5.8 Refresh Token Table

Stores refresh tokens used for authentication purposes, with expiration times.

- **Primary Key:** token_id
- **Foreign Key:** user_id (linked to the `users` table)
- **Attributes:**
  - `token_id`: Unique identifier for the refresh token.
  - `expiration_time`: Expiration time of the refresh token.
  - `refresh_token`: The refresh token used for authentication.
  - `user_id`: Foreign key referring to the user (linked to the `users` table).

5.9 User Orders Table

Stores information about user orders, including the course ordered, amount, and status.

- **Primary Key:** order_id
- **Foreign Key:** user_id (linked to the `users` table)
- **Attributes:**
  - `order_id`: Unique identifier for each order.
  - `amount`: Total amount for the order.
  - `course`: The course or product that was ordered.
  - `email`: Email of the user who made the order.
  - `name`: Name of the user who made the order.
  - `order_status`: Status of the order (e.g., pending, completed).
  - `phone`: Phone number of the user who made the order.
  - `razorpay_order_id`: The Razorpay order ID associated with the payment.

Chapter 6

API Documentation

6.1 Product Service

6.1.1 POST - Product file upload

```
URL: http://localhost:8080/product/file/upload
Method: POST
Body: form-data
File: <file>
```

6.1.2 POST - Create Product

```
URL: http://localhost:8080/api/product/add-product
Method: POST
Authorization: Bearer <token>
Body: form-data
ProductDto:
{
    "productId": 10,
    "title": "TESTING Data Science with Python",
    "description": "Analyze data and build machine learning models using Python lib
    "category": "Data Science",
    "price": 99.99,
    "rating": 4.7,
    "buyers": ["alice.smith@example.com", "bob.jones@example.com"],
    "productImage": "default.png",
    "productUrl": "url"
}
```

6.1.3 GET - Get Product by ID

```
URL: http://localhost:8080/api/product/4
Method: GET
Authorization: Bearer <token>
```

6.1.4 GET - Get All Products

```
URL: http://localhost:8080/api/product/all
Method: GET
Authorization: Bearer <token>
```

6.1.5 PUT - Update Product

```
URL: http://localhost:8080/api/product/update/1
Method: PUT
Authorization: Bearer <token>
Body: form-data
ProductDtoObj:
{
    "productId": 1,
    "title": "UPDATED Machine Learning A-Z",
    "description": "Hands-on Machine Learning course covering Python and R.",
    "category": "Development",
    "price": 49.99,
    "rating": 4.8,
    "buyers": ["john.doe@example.com", "jane.doe@example.com"],
    "productImage": "default.png",
    "productUrl": "url"
}
```

6.1.6 DELETE - Delete product by ID

```
URL: http://localhost:8080/api/product/delete/7
Method: DELETE
Authorization: Bearer <token>
```

6.2 Authentication Service

6.2.1 POST - Verify Mail

```
URL: http://localhost:8080/forgotPassword/verifyMail/example@gmail.com
Method: POST
```

6.2.2 POST - Verify OTP

```
URL: http://localhost:8080/forgotPassword/verifyOtp/111111/example@gmail.com
Method: POST
```

6.2.3 POST - Change Password

```
URL: http://localhost:8080/forgotPassword/changePassword/example@gmail.com
Method: POST
Body: raw (json)
{
```

```
    "password": "12345678",
    "repeatPassword": "12345678"
}
```

6.2.4 POST - Register New User

```
URL: http://localhost:8080/api/auth/register
Method: POST
Body: raw (json)
{
    "name": "John",
    "email": "john@gmail.com",
    "username": "_john_01",
    "password": "12345678"
}
```

6.2.5 POST - Login User

```
URL: http://localhost:8080/api/auth/login
Method: POST
Body: raw (json)
{
    "email": "john@gmail.com",
    "password": "12345678"
}
```

6.2.6 POST - Refresh Token

```
URL: http://localhost:8080/api/auth/refresh
Method: POST
Body: raw (json)
{
    "refreshToken": "ed9cd08d-9c72-4791-8f30-b5b46fc0b5cc"
}
```

6.2.7 GET - Get User currently logged in

```
URL: http://localhost:8080/api/auth/me
Method: GET
Authorization: Bearer <token>
```

6.3 User Service

6.3.1 GET - Get All users

```
URL: http://localhost:8080/api/users
Method: GET
Authorization: Bearer <token>
```

6.3.2 GET - Get user by ID

```
URL: http://localhost:8080/api/users/3
Method: GET
Authorization: Bearer <token>
```

6.3.3 DELETE - Delete user by ID

```
URL: http://localhost:8080/api/users/3
Method: DELETE
Authorization: Bearer <token>
```

6.4 Blogs Service

6.4.1 POST - Blog file upload

```
URL: http://localhost:8080/blog/file/upload
Method: POST
Authorization: Bearer <token>
Body: form-data
File: <file>
```

6.4.2 POST - Create Blog

```
URL: http://localhost:8080/api/blog/add-blog
Method: POST
Authorization: Bearer <token>
Body: form-data
BlogDto:
{
    "userId": 1,
    "title": "SECOND Blog Title",
    "content": "This is a sample SECOND blog content.",
    "category": "Technology"
}
```

6.4.3 GET - Get Blog by ID

```
URL: http://localhost:8080/api/blog/45
Method: GET
Authorization: Bearer <token>
```

6.4.4 GET - Get All Blogs

```
URL: http://localhost:8080/api/blog/all
Method: GET
Authorization: Bearer <token>
```

6.4.5 PUT - Update Blog

```
URL: http://localhost:8080/api/blog/update/14
Method: PUT
Authorization: Bearer <token>
Body: form-data
BlogDtoObj:
{
    "userId": 1,
    "title": "UPDATED Blog Title",
    "content": "This is a sample blog content.",
    "category": "Technology"
}
```

6.4.6 DELETE - Delete Blog by ID

```
URL: http://localhost:8080/api/blog/delete/24
Method: DELETE
Authorization: Bearer <token>
```

6.5 Contact Service

6.5.1 GET - Get All Contacts

```
URL: http://localhost:8080/api/contacts
Method: GET
Authorization: Bearer <token>
```

6.5.2 POST - Post a new Contact

```
URL: http://localhost:8080/api/contacts
Method: POST
Authorization: Bearer <token>
Body: raw (json)
{
    "userId": 4,
    "name": "John Doe",
    "email": "john@gmail.com",
    "subject": "Inquiry",
    "message": "Hello, I have a question."
}
```

6.5.3 GET - Get Contact by ID

```
URL: http://localhost:8080/api/contacts/8
Method: GET
Authorization: Bearer <token>
```

6.5.4 PUT - Update Contact

```
URL: http://localhost:8080/api/contacts/2
Method: PUT
Authorization: Bearer <token>
Body: raw (json)
{
    "userId": 4,
    "name": "Jane Smith",
    "email": "jane.smith@example.com",
    "subject": "Updated Inquiry",
    "message": "Updated inquiry message"
}
```

6.5.5 DELETE - Delete Contact by ID

```
URL: http://localhost:8080/api/contacts/1
Method: DELETE
Authorization: Bearer <token>
```

6.5.6 GET - Get all contacts of particular user

```
URL: http://localhost:8080/api/contacts/user/4
Method: GET
Authorization: Bearer <token>
```

6.6 Feedback Service

6.6.1 GET - Get All Feedbacks

```
URL: http://localhost:8080/api/feedbacks
Method: GET
Authorization: Bearer <token>
```

6.6.2 POST - Post a new Feedback

```
URL: http://localhost:8080/api/feedbacks
Method: POST
Authorization: Bearer <token>
Body: raw (json)
{
    "userId": 4,
    "name": "John Doe",
    "email": "john.doe@example.com",
    "phone": "123-456-7890",
    "message": "Great service, I am very satisfied!"
}
```

6.6.3 GET - Get Feedback by ID

```
URL: http://localhost:8080/api/feedbacks/8
Method: GET
Authorization: Bearer <token>
```

6.6.4 PUT - Update Feedback

```
URL: http://localhost:8080/api/feedbacks/1
Method: PUT
Authorization: Bearer <token>
Body: raw (json)
{
    "userId": 4,
    "name": "John Doe",
    "email": "john.doe@example.com",
    "phone": "987-654-3210",
    "message": "Updated feedback message!"
}
```

6.6.5 DELETE - Delete Feedback by ID

```
URL: http://localhost:8080/api/feedbacks/1
Method: DELETE
Authorization: Bearer <token>
```

6.6.6 GET - Get all feedbacks of particular user

```
URL: http://localhost:8080/api/feedbacks/user/4
Method: GET
Authorization: Bearer <token>
```

6.7 Order Service

6.7.1 GET - Get all orders of user

```
URL: http://localhost:8080/api/orders/user/1
Method: GET
Authorization: Bearer <token>
```

6.7.2 GET - Get All Orders

```
URL: http://localhost:8080/api/orders
Method: GET
Authorization: Bearer <token>
```

6.7.3 GET - Get Order by ID

```
URL: http://localhost:8080/api/orders/4
Method: GET
Authorization: Bearer <token>
```

Chapter 7

Testing

7.1 Tools Used

7.1.1 Postman

Postman is a popular tool used for testing RESTful API endpoints. It is highly effective for verifying the functionality of backend services by sending HTTP requests and inspecting the responses. Postman allows developers to create and manage various API requests, including GET, POST, PUT, DELETE, etc. This helps in testing different functionalities, such as authentication, CRUD operations, product management, user management, and order processing in the Edu Consultancy platform.

Through Postman, we can verify if the backend APIs are returning the expected status codes, response bodies, and headers. Additionally, it is useful in testing edge cases, error responses, and validating authentication/authorization mechanisms like JWT tokens. Postman also supports the automation of tests, which can be run across multiple environments or services, ensuring that the backend is functioning properly after each change or deployment.

7.1.2 Browser DevTools

Browser Developer Tools, often referred to as DevTools, are built-in tools available in modern web browsers like Chrome, Firefox, and Edge. These tools allow developers to inspect and debug the behavior of the frontend application. Browser DevTools help in monitoring network requests, analyzing responses, and debugging JavaScript issues, which are crucial for ensuring the correct functioning of the frontend.

In the Edu Consultancy platform, Browser DevTools can be used to verify if the frontend is correctly interacting with the backend APIs. By inspecting the network tab, developers can ensure that the frontend is sending the right data in API requests and receiving the expected responses from the backend. This also helps in debugging issues like failed API calls, incorrect data being sent or received, and UI rendering errors. DevTools also offer performance insights, which can be used to optimize the frontend.

7.1.3 Test Cases for Backend Services

The backend of the Edu Consultancy platform is built using Spring Boot and tested with **JUnit** and **Mockito**. JUnit helps in writing unit tests to verify the cor-

rectness of business logic, ensuring that services like user registration, login, product management, and order processing behave as expected. For example, JUnit tests validate user registration, password reset (including OTP generation), and product purchase logic.

Mockito is used to mock dependencies like databases and external services, allowing tests to focus solely on the business logic. This is especially useful for services such as the feedback and contact services, where Mockito can mock repositories, enabling tests to verify feedback processing and message storage without actual database interactions. By isolating the service logic, Mockito helps speed up testing and catches bugs early in the development cycle.

7.1.4 Summary of Testing Strategy

In summary, the testing of the Edu Consultancy platform is performed using a combination of tools and techniques:

- **Postman** is used for testing the RESTful APIs, validating the responses, and ensuring the backend functionality is working as expected.
- **Browser DevTools** are used to debug and inspect frontend interactions with backend services, making sure the frontend correctly communicates with the backend and displays the expected results.
- **JUnit** is used to write unit tests for backend services, ensuring that the business logic is correct and performs as expected under different conditions.
- **Mockito** is used to mock dependencies and isolate services, ensuring that backend logic is tested independently of external systems like databases and APIs.

This approach ensures a comprehensive testing strategy, validating both the frontend and backend components of the Edu Consultancy platform to guarantee functionality, security, and user experience.

Chapter 8

Future Scope and Enhancements

8.1 AI-Based Consultancy Matching

Description:
Integrate artificial intelligence to match users with consultants based on their profiles and preferences. Machine learning algorithms can analyze historical data to recommend the most suitable consultants.

Proposed Features:

- **Profile-Based Matching:** Matches consultants to users based on academic goals and feedback history.
- **Recommendation System:** Implements collaborative or content-based filtering techniques.
- **Dynamic Scoring:** Dynamically ranks consultants based on recent feedback.

Benefits:

- Personalized recommendations improve user satisfaction.
- Reduces the time needed to find suitable consultants.
- Enhances user retention through tailored suggestions.

Implementation Steps:

1. Collect user preferences and feedback data.
2. Train a recommendation model using libraries like TensorFlow.
3. Deploy the model as a microservice accessible via REST APIs.

8.2 Analytics Dashboard for Admins

Description:
Develop an interactive dashboard to visualize key metrics, trends, and user feedback.

Proposed Features:

- **Real-Time Data Display:** Visualizes active users, subscriptions, and revenue.

- **Visualization Tools:** Dynamic charts and graphs using Chart.js or D3.js.
- **Customizable Reports:** Export insights as PDFs or CSVs.
- **Feedback Trends:** Analyze user feedback for actionable insights.

Benefits:

- Enables data-driven decision-making.
- Identifies trends and opportunities for growth.
- Improves operational efficiency by centralizing metrics.

Implementation Steps:

1. Aggregate data using backend APIs.
2. Design a responsive UI with React.
3. Utilize a visualization library for interactive charts.

8.3 Mobile App Integration

Description:
Expand the platform's reach by developing a mobile application to improve accessibility and engagement.

Proposed Features:

- **Comprehensive Access:** Registration, browsing, and subscription management on mobile.
- **Push Notifications:** Updates and reminders for users.
- **Offline Capabilities:** Save content for offline use.
- **Cross-Platform Compatibility:** Build with React Native or Flutter.

Benefits:

- Attracts mobile-first users.
- Increases user engagement with accessibility features.
- Promotes retention through timely notifications.

Implementation Steps:

1. Design an intuitive mobile UI.
2. Reuse existing REST APIs for backend integration.
3. Implement push notifications using Firebase.

Chapter 9

Conclusion

The **Edu Consultancy** project successfully integrates modern web technologies to provide an efficient and user-friendly platform for both users and administrators. By leveraging a robust tech stack comprising React for the frontend and Spring Boot for the backend, the system ensures a seamless experience for its users. Features such as secure authentication, intuitive subscription management, and powerful CRUD operations empower users and administrators alike.

The integration of payment processing through Razorpay and email-based notifications adds value and enhances the overall usability of the platform. Moreover, the project is designed with scalability and extensibility in mind, making it well-suited for future enhancements such as AI-based consultancy matching, analytics dashboards, and mobile app integration.

This project demonstrates the practical application of full-stack development principles and highlights the potential for further innovation in the education consultancy domain. With additional enhancements, the platform can evolve into a comprehensive solution for educational consultancy services, catering to diverse user needs and ensuring long-term growth and sustainability.