UNIVERSITY INSTITUTE OF ENGINEERING

Java Project

Online Student Feedback & Resolution System.

Subject Code – 23CSP-339

Submitted To:
Faculty Name: Er. Deep Prakash
(E18479) UID: 23BCS11346

Submitted By:
Name: Ronak Jain
23BCS10225
Section: KRG - 2B
Semester: 5th

Module 1: User Authentication and Roles

This initial module is the security foundation for the entire project. My goal here isn't just to create a login page, but to build a robust system that correctly identifies users as either a 'student' or an 'admin' and uses that role to control what they can see and do. This directly ties into my plan for role-based access.

Backend Plan: Spring Boot

The backend will be the authority on user identity and permissions.

- **Project Setup:** I'll start by using the Spring Initializr to generate a new Spring Boot project. I'll be sure to include key dependencies: Spring Web for creating REST APIs, Spring Security for the authentication framework, Spring Data JPA to interact with the database, and the PostgreSQL Driver.
- **Database and Entity:** I will configure the application.properties file to connect to my local PostgreSQL instance. Then, I'll create a Java class called User and annotate it as a JPA entity. This class will directly map to the
  users table schema I designed: id, name, email, role, and created_at. I'll also create a UserRepository interface that extends JpaRepository to handle database operations like finding a user by their email.
- **API Endpoints:** I'll create a controller to handle authentication requests. The main endpoint will be a POST request to a path like /api/auth/login. This endpoint will accept a user's email and password.
- **Security Implementation (The Core Logic):** This is the most critical part. I plan to use **Spring Security and JSON Web Tokens (JWT)** for a stateless authentication system.
    1. When a user submits their credentials to the /login endpoint, Spring Security will first validate them against the user data in my PostgreSQL database.
    2. If the credentials are correct, the backend will generate a JWT. This token is a secure string that will contain essential user information, specifically their user ID and their
       role (e.g., 'student' or 'admin').
    3. This JWT is then sent back to the frontend in the response.
    4. For all future requests to protected parts of the app, the frontend will need to include this JWT in the Authorization header. I'll configure a Spring Security filter that intercepts every incoming request, validates the token, and extracts the user's role to authorize their action.

Frontend Plan: React

The frontend will be responsible for the user-facing login experience and for managing the session.

- **UI Components:** I'll build a Login.js component that contains a simple form with fields for email and password, along with a submit button. I'll also add basic form validation to ensure the fields aren't empty.
- **API Service:** I'll create a separate service file, perhaps AuthService.js, to manage all communication with the backend's authentication endpoints. This keeps my component code clean and organized.
- **State Management:** As I noted in my project overview, I'll need to manage the user's session state. After a successful login, the AuthService will receive the JWT from the backend. I plan to use **React's Context API** to store this token and the user's role in a global state. This makes the user's authentication status available to any component in the app. I'll also save the token in the browser's localStorage so the user remains logged in even after a page refresh.
- **Protected Routes:** Using the react-router-dom library, I'll create a "Protected Route" component. This component will check the global authentication state. If a user is not logged in, it will redirect them to the login page. Furthermore, it will check the user's role to ensure a student cannot access admin-only pages (like /admin/dashboard) and vice versa.

By the time I finish this module, I will have a complete and secure authentication flow. A user will be able to log in, have their session managed on the frontend, and be routed to the correct dashboard based on a role that is securely verified by the backend on every request.

Module 2: Student Feedback Submission (Detailed Plan)

This module is the heart of the student-facing side of the application. Its purpose is to provide a simple and secure way for students to voice their opinions. My focus will be on creating a user-friendly form that captures structured feedback and respects the user's choice to remain anonymous.

Frontend Plan: React

The user's entire submission experience will be handled here. I'll need to build a dynamic and responsive form that communicates smoothly with the backend.

- **Component Design:** I'll create a dedicated React component, let's call it FeedbackForm.js. This component will be responsible for rendering the form fields and managing their state.
- **Form Fields and State:** To make the feedback structured and easy to analyze later , the form will contain several input types. I'll use the useState hook in React to manage the data for each:
    1. A main textarea for the detailed feedback content.
    2. A star-rating component (from 1 to 5) for a quick quantitative measure.
    3. A dropdown menu for categorizing the feedback (e.g., "Facilities," "Courses," "Campus Life").
    4. Most importantly, a single checkbox labeled **"Submit Anonymously"**. Its boolean state will be crucial.
- **Validation:** I'll implement client-side validation to ensure data quality before it even leaves the browser. For example, the main content area will be a required field. If a user tries to submit an empty form, an error message will appear, guiding them to fill it out correctly.
- **API Interaction:** I'll create a FeedbackService.js file to handle the API call. When a student clicks the "Submit" button:
    1. A function in this service will gather all the state data from the form into a single JSON object.
    2. It will then send this object via a secure POST request to the backend REST API. The JWT authentication token from Module 1 will be included in the request header to identify the logged-in student.

Backend Plan: Spring Boot

The backend will receive the submitted data, validate it, and store it reliably in the database.

- **API Endpoint:** I'll create a FeedbackController.java with a protected POST endpoint, something like /api/feedback/submit. Spring Security will ensure that only an authenticated user can access this endpoint.
- **Data Handling:** I will create a FeedbackRequestDTO (Data Transfer Object) class. This class will define the structure of the incoming JSON data from the React form, making the code cleaner and more secure.
- **Service Layer Logic:** The controller will pass the DTO and the student's ID (extracted from the JWT) to the FeedbackService. This is where the core business logic will reside. The service will perform these key actions:
    1. **Server-Side Validation:** It will re-validate the incoming data to ensure its integrity.
    2. **Entity Creation:** It will create a new Feedback entity object.

3. **Set Initial Values:** It will set the feedback's initial status to 'open' and set the created_at timestamp.
4. **Handle Anonymity:** It will check the is_anonymous boolean from the DTO. If it's true, it will set the corresponding flag on the Feedback entity. Although the system knows the student_id, this flag will tell the backend to hide this information from admins later on.

- **Database Storage:** Finally, the FeedbackService will use the FeedbackRepository to save the new Feedback entity into my PostgreSQL feedback table.

Once this module is complete, a logged-in student will be able to fill out a detailed form, choose to remain anonymous, and have their feedback securely stored in the database, ready for an admin to review.

Module 3: Student Dashboard & Feedback Tracking

After a student submits feedback, they need a way to see what happens next. This module is all about transparency and closing the loop. I'll build a personal dashboard where a student can track the status of their submissions and view any communication from the administration. This ensures they feel heard and remain engaged in the resolution process.

Backend Plan: Spring Boot

The backend's primary job here is to serve the student's data securely and efficiently, ensuring they can only see their own feedback.

- **API Endpoints:** I'll expand my FeedbackController with two new protected GET endpoints:
  - /api/feedback/my-feedback: This endpoint will fetch a list of all feedback submissions made by the currently logged-in student.
  - /api/feedback/{feedbackId}: This endpoint will retrieve the full details for a single feedback entry, including its threaded discussion.
- **Security and Data Logic:** This is where the role-based visibility I planned comes into play. When a request hits /api/feedback/my-feedback, my service layer will extract the student_id from the JWT. It will then query the feedback table and explicitly filter the results to return *only* the records matching that student_id. This is a critical security step to maintain privacy.
- **Data Transfer Objects (DTOs):** I won't send the raw database entities to the frontend. Instead, I'll create specific DTOs:

   - A
     FeedbackSummaryDTO for the list view, containing just the essential info like an ID, a snippet of the content, and its current status ('open', 'in_progress', 'resolved').
   - A FeedbackDetailDTO for the detailed view. This will include the full feedback content, its status, and a list of
     CommentDTOs to represent the threaded discussion. This data will be pulled from the
     feedback_thread table I designed.

Frontend Plan: React

The frontend will translate the data from the backend into a clear and interactive user interface. This is where I'll focus on state management to ensure the UI updates in real-time as new information becomes available.

- **Component Structure:** I'll build a few new components:
  1. StudentDashboard.js: This will be the main container page, protected by my routing logic from Module 1.
  2. FeedbackList.js: This component will be rendered inside the dashboard. It will receive the list of feedback summaries from the API and display them, perhaps as a series of cards or table rows. Each entry will clearly show the feedback status, maybe with a color-coded tag.
  3. FeedbackDetail.js: When a user clicks an item in the FeedbackList, they'll be navigated to this component. It will display the original feedback in full, followed by a chronological list of responses from admins, creating that "conversation thread" feel I'm aiming for.
- **State Management and API Calls:**
  1. When the StudentDashboard component first loads, I'll use a useEffect hook to trigger a function in my FeedbackService.
  2. This service will make the GET request to the /api/feedback/my-feedback endpoint.
  3. The array of feedback summaries returned from the API will be stored in the component's state using useState.
  4. React will automatically re-render the FeedbackList component with the fetched data, allowing the student to see their submission history.

By completing this module, I'll have a fully functional feedback loop for students. They can submit an issue in Module 2 and track its entire lifecycle here, from "open" to "resolved," including any back-and-forth communication with the administration.

Module 4: Admin Dashboard & Feedback Viewing

This module is the command center for administrators, moderators, and faculty. Unlike the student-facing modules, this one is about providing a high-level, comprehensive view of all the feedback within the system. The goal is to give admins the tools they need to efficiently review, filter, and understand feedback trends at a glance.

Backend Plan: Spring Boot

The backend needs to provide powerful and secure endpoints that can serve a large amount of data and aggregate it for analysis.

- **API Endpoints:** I'll create a new AdminController or add admin-specific endpoints to my existing controllers. These endpoints will be the backbone of the dashboard:
  - GET /api/admin/feedback: This will be the main endpoint to fetch a list of **all feedback entries** in the system. To make this functional, it will need to support query parameters for filtering, such as ?status=open or ?department=CSE. My service layer will dynamically build database queries based on these parameters.
  - GET /api/admin/analytics: A separate endpoint designed to provide aggregated data for the charts on the dashboard. It will return a JSON object with summarized data, like the total count of feedback per status or category.
- **Security:** This is non-negotiable. Every endpoint designed for this module will be strictly protected using Spring Security. Before any code in the controller is executed, the security filter will validate the user's JWT and ensure their role is 'admin'. If a student attempts to access these endpoints, they will receive a "Forbidden" error.
- **Data Handling and Anonymity:** When the backend fetches feedback for an admin, it will pull all records from the database. However, it will still respect the is_anonymous flag I designed in the schema. My service layer will include logic to check this flag; if it is true, the DTO sent to the frontend will have the student's identifying information removed or nulled out to maintain the promise of anonymity.

Frontend Plan: React

The frontend will be a data-rich Single Page Application (SPA) that provides a clear and interactive interface for managing feedback.

- **Component Structure:** I'll build the dashboard using a modular approach:
  - AdminDashboard.js: The main container page, which will be a protected route accessible only to admins.

    - FeedbackTable.js: This component will display all feedback submissions in a clear, organized table. Columns will include details like a unique ID, content snippet, status, and submission date. I'll make the table sortable by column.
    - FilterControls.js: A dedicated component containing dropdown menus and date pickers. When an admin changes a filter, this component will update the state, triggering a new API call to fetch the filtered data.
    - AnalyticsView.js: This component will focus on data visualization. It will fetch data from the analytics endpoint and render it into meaningful charts.
- **Analytics and Visualization:** As I planned in my project overview, this is where I'll get to work with a data visualization library. My plan is to use
 Chart.js within the AnalyticsView component. I'll use it to create visuals that show trends and provide quick insights, such as:
    - A pie chart breaking down feedback by status (open, in_progress, resolved).
    - A bar chart showing the most common feedback categories.
- **State Management:** The AdminDashboard will fetch the initial, unfiltered list of feedback when it first loads using a useEffect hook. This data will be stored in the component's state. When an admin applies a filter, the state for the filter parameters will change, which will trigger a new API call and update the feedback list in the UI.

By completing this module, I will have a functional and insightful dashboard where administrators can see the entire landscape of student feedback, filter it down to what's important, and understand trends through visual charts.

Module 5: Feedback Management & Resolution

This is the final, action-oriented module for the administration. While Module 4 was about viewing and analyzing feedback, this module is about *acting* on it. Here, I'll build the tools that allow admins to update the status of feedback, communicate directly with students through threaded replies, and officially resolve issues with full accountability.

Backend Plan: Spring Boot

The backend for this module will handle state changes, create new data entries for replies and logs, and ensure every action is authorized and recorded.

- **API Endpoints:** I'll create a set of secure endpoints in my AdminController to handle the management actions:
    - PUT /api/admin/feedback/{feedbackId}/status: An endpoint that allows an admin to change the status of a feedback entry. It will accept a request body with the new status, such as 'in_progress' or 'resolved'.

    - POST /api/admin/feedback/{feedbackId}/reply: This endpoint will allow an admin to post a reply to a specific feedback thread. The request body will contain the comment text.
    - POST /api/admin/feedback/{feedbackId}/resolve: A dedicated endpoint for marking feedback as resolved. It will take a final resolution_note in the request body, update the feedback's status, and create a log entry.
- **Security:** As with all admin functionalities, these endpoints will be strictly protected by Spring Security. The system will verify the user's JWT to ensure they have the 'admin' role before allowing any of these update or creation operations to proceed.
- **Service Layer Logic and Database Interaction:** My FeedbackService will implement the core business logic for these actions:
    - **Status Updates:** The service will find the feedback by its ID and update the value in the status column of the feedback table.
    - **Threaded Replies:** When an admin submits a reply, the service will create a new record in the feedback_thread table. This record will be linked to the parent feedback_id and will store the admin's user_id in the created_by field, along with a timestamp, to maintain a clear conversation history. This enables the back-and-forth discussion I envisioned.
    - **Resolution Logging:** This is a key feature for accountability. When the resolve endpoint is called, the service will perform two critical database operations: first, it updates the feedback's status to 'resolved' , and second, it creates a new, immutable record in the resolution_log table. This log will capture who resolved the issue (resolved_by), the final note, and the exact timestamp.

Frontend Plan: React

The frontend for this module will integrate directly into the admin's detailed feedback view, providing a seamless interface for managing the issue's lifecycle.

- **Component Integration:** I will build these interactive elements into an AdminFeedbackDetail.js component. An admin will navigate here after clicking on a specific feedback item from the table I built in Module 4.
- **Interactive UI Elements:** This component will feature:
    - A **Status Dropdown** pre-filled with the current status. An admin can select a new status ('In Progress', 'Resolved'), which will trigger a PUT request to the backend.
    - A **Reply Form** with a textarea and a "Post Reply" button. This will allow admins to add comments to the threaded discussion.

    - A **Resolution Section** that includes a "Finalize Resolution" button. Clicking this might open a modal where the admin can type their final resolution_note before submitting, which triggers the final API call to resolve the issue.
- **Real-Time Updates:** I'll use React's state management to provide a responsive experience. After an admin posts a reply or changes a status, the component will automatically re-fetch the latest data for that feedback item. This will instantly update the UI to show the new comment in the thread or the changed status tag, giving the admin immediate confirmation of their action.

By completing this module, the system will have a complete workflow. An admin can view all feedback (Module 4), dive into a specific issue, communicate with the student, update the issue's status, and formally resolve it, with every significant action logged for future reference.