Full Stack Project: Online Student Feedback & Resolution System

Subject Code: 23CSP-339 **Student:** Ronak Jain (23BCS10225)

Module 5: Feedback Management & Resolution

1. Objective

This is the final, action-oriented module that completes the project's entire workflow. While Module 4 provided the "READ" capability for admins, Module 5 implements the "UPDATE" functionality. The objective was to build the tools for an admin to act on feedback by changing its status, posting replies to create a conversation, and formally resolving issues. A key goal was also to "close the loop" by displaying these admin replies on the student's dashboard.

2. Technologies Used

- **Backend:** Spring Boot, Spring Security, Spring Data MongoDB
- **Frontend:** React (using `useParams`, `useNavigate`, `useEffect`, `useState`), Axios, React Router
- **Database:** MongoDB

3. Backend Implementation (Spring Boot)

The backend API was expanded with new endpoints to handle specific management actions on individual feedback documents.

3.1. DTOs (`dto` package)

Two new DTOs were created to handle incoming admin actions:

- **`StatusDTO.java`:** A simple DTO with a single `status` field, used for validation when an admin updates a feedback item's status.
- **`ReplyDTO.java`:** A DTO with a `content` field, used to validate the text from an admin's reply.

3.2. Service Layer (`service` package)

- **`FeedbackService.java`:** This service was updated with the core business logic for all admin actions:

    - `getFeedbackByIdForAdmin(String feedbackId)`: Fetches a single `Feedback` document and merges it with the student's details (respecting anonymity) to create the `FeedbackAdminViewDTO`.
    - `updateFeedbackStatus(String feedbackId, String newStatus)`: Finds the feedback document by its ID, sets its `status` field to the new value, and saves it.
    - `postReplyToFeedback(String feedbackId, ReplyDTO replyDTO, String adminUserId)`: This is a key method. It:
        1. Finds the feedback document by its ID.
        2. Creates a new `Comment` object, populating it with the admin's reply content and their `adminUserId`.
        3. Adds this new `Comment` to the `thread` list embedded within the `Feedback` document.
        4. **Business Logic:** It automatically updates the feedback's status to `"in_progress"` if it was previously `"open"`.
        5. Saves the entire updated `Feedback` document back to MongoDB.

3.3. API Endpoint (`controller` package)

- **`AdminController.java`:** This controller was expanded with three new, secure endpoints:
    - `GET /api/admin/feedback/{id}`: Uses `@PathVariable` to get the feedback ID from the URL. It calls the `getFeedbackByIdForAdmin` service method to return the detailed data for a single ticket.
    - `PUT /api/admin/feedback/{id}/status`: Receives a `StatusDTO` and calls the `updateFeedbackStatus` service method to update the item.
    - `POST /api/admin/feedback/{id}/reply`: Receives a `ReplyDTO` and uses the injected `Authentication` object to securely get the logged-in admin's `User` object. It then passes the reply and the admin's ID to the `postReplyToFeedback` service method.

4. Frontend Implementation (React)

The frontend was completed by building the final detail page and connecting all the components.

4.1. Service Layer (`services` package)

- **`AdminService.js`:** This service was updated with new functions to match the backend API:
    - `getFeedbackById(id)`: Calls the `GET .../{id}` endpoint.
    - `updateFeedbackStatus(id, status)`: Calls the `PUT .../{id}/status` endpoint.
    - `postReply(id, content)`: Calls the `POST .../{id}/reply` endpoint.

4.2. UI Components (`components` package)

- **`AdminDashboard.js` (Updated):** The "View" button in the feedback table was activated. An `onClick` handler (`handleViewClick`) was added that uses the `useNavigate` hook to redirect the admin to the new detail page, passing the feedback's ID in the URL.
- **`AdminFeedbackDetail.js` (New Component):** This is the primary UI for this module.
    - It uses the `useParams` hook to get the feedback `id` from the URL.
    - A `useEffect` hook calls `AdminService.getFeedbackById(id)` on page load to fetch the item's details.
    - `handleStatusChange`: An event handler for the status dropdown that calls `AdminService.updateFeedbackStatus` and then updates the *local state* with the response, causing the UI to update instantly.
    - `handleReplySubmit`: An event handler for the reply form that calls `AdminService.postReply` and, on success, updates the local state with the new `thread` and clears the text box.
- **`AdminFeedbackDetail.css` (New File):** A dedicated CSS file was created to style this new page, using a two-column layout for "Details" and "Admin Actions" and adding styles for the "Conversation Thread."

4.3. Routing (`App.js`)

- The main router file was updated with the new dynamic route:
    - `<Route path="/admin/feedback/:id" ... />`: This route renders the `AdminFeedbackDetail` component and is wrapped in the `ProtectedRoute` to ensure only admins can access it.

4.4. Closing the Loop: Updating the Student Dashboard

- A critical part of this module was updating the student's view.
- **`FeedbackList.js` (Updated):** This component (on the student's dashboard) was modified to check if `item.thread` exists and has items. If it does, it now maps over the `thread` array and renders each admin reply in a "Conversation Thread" section within the feedback card.
- **`FeedbackList.css` (Updated):** Styles were added to make the admin replies visually distinct.