Full Stack Project: Online Student Feedback & Resolution System

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

Module 4: Admin Dashboard & Feedback Viewing

1. Objective

With the student-facing "Create" and "Read" operations complete, the objective of Module 4 was to build the administrator's "command center." This module focuses on providing a secure, high-level dashboard for administrators to view, filter, and analyze *all* feedback submitted to the system. Key goals included implementing analytics (charts) as planned, ensuring student anonymity was respected, and providing tools to filter the data.

2. Technologies Used

- **Backend:** Spring Boot, Spring Security, Spring Data MongoDB (using `@Aggregation`)
- **Frontend:** React (using `useEffect`, `useState`, `useMemo`), Axios, React Router, **Chart.js** (`react-chartjs-2`)
- **Database:** MongoDB

3. Backend Implementation (Spring Boot)

The backend was significantly expanded to provide powerful, admin-only API endpoints for data retrieval and aggregation.

3.1. DTOs (`dto` package)

Two new DTOs were created to shape the data for the admin dashboard:

- **`FeedbackAdminViewDTO.java`:** This DTO was created to send a "merged" object to the frontend, combining the full `Feedback` document with relevant details from the `User` document (like `studentName` and `studentEmail`).
- **`AnalyticsDTO.java`:** A simple DTO created to hold the results of our aggregation queries. It contains `_id` (the value grouped by, e.g., "open") and `count`.

3.2. Repository (`repository` package)

- **`FeedbackRepository.java`:** This interface was updated to include powerful aggregation queries, as specified in the project plan.

    - `List<AnalyticsDTO> countByStatus()`: Uses the `@Aggregation` annotation to define a MongoDB pipeline that groups all feedback by the `status` field and returns the count for each.
    - `List<AnalyticsDTO> countByCategory()`: Uses `@Aggregation` to group all feedback by the `category` field and return the counts.

3.3. Service Layer (`service` package)

- **`FeedbackService.java`:** This service was enhanced with new methods to serve the admin dashboard:
    - `getAllFeedbackForAdmin()`: This is the module's core logic. It performs an efficient, N+1-safe data fetch:
        1. Fetches *all* feedback from `FeedbackRepository`.
        2. Extracts all unique `studentId`s from the list.
        3. Fetches all corresponding `User` documents in a *single* database call.
        4. Places these users into a `Map<String, User>` for instant lookups.
        5. Iterates the feedback list, checking the `isAnonymous` flag. If `true`, it populates the DTO with "Anonymous". If `false`, it looks up the student's details in the map.
        6. Returns a `List<FeedbackAdminViewDTO>`.
    - `getStatusAnalytics()` & `getCategoryAnalytics()`: These methods simply call the new aggregation methods on the `FeedbackRepository`.

3.4. API Endpoint (`controller` package)

- **`AdminController.java` (New Controller):** A new, dedicated controller was created for all admin functionality.
    - **Security:** It is annotated with `@RequestMapping("/api/admin")`. Due to the `SecurityConfig` (from Module 1), all endpoints within this controller are automatically protected and require `ROLE_ADMIN`.
    - `GET /api/admin/feedback`: Calls the `getAllFeedbackForAdmin()` service method and returns the full list of feedback.
    - `GET /api/admin/analytics`: Calls both analytics service methods and returns their data in a single, combined JSON object (`Map.of("statusData", ..., "categoryData", ...)`), saving the frontend from making two separate API calls.

4. Frontend Implementation (React)

The frontend `AdminDashboard` was built out from a placeholder into a feature-rich Single Page Application (SPA).

4.1. Installation

- The `chart.js` and `react-chartjs-2` libraries were installed via `npm` to provide data visualization capabilities.

4.2. Service Layer (`services` package)

- **`AdminService.js` (New File):** A new, dedicated service was created for all admin API calls.
    - It uses the `authHeader()` helper (from `AuthService`) to ensure the admin's JWT is attached to all requests.
    - `getAllFeedback()`: Calls the `GET /api/admin/feedback` endpoint.
    - `getAnalytics()`: Calls the `GET /api/admin/analytics` endpoint.

4.3. UI Components (`components` package)

- **`AdminDashboard.js` (Updated):** This component was completely rebuilt to be the main admin UI.
    - **Chart.js Registration:** It correctly imports and *registers* the necessary components from `Chart.js` (e.g., `ArcElement`, `CategoryScale`, `BarElement`).
    - **Data Fetching:** A `useEffect` hook calls `Promise.all` to fetch both feedback and analytics data in parallel on component load. `loading` and `error` states are managed.
    - **Data Formatting:** The `useMemo` hook is used to format the raw data from the `getAnalytics()` call into the specific data structures required by the `Pie` and `Bar` chart components. This optimizes performance by preventing re-calculation on every render.
    - **Filtering:** `useState` is used to manage the state of the "Status" and "Category" filter dropdowns.
    - **`filteredFeedback`:** Another `useMemo` hook is used to create the list of feedback to be rendered. It takes the master `feedbackList` and applies the `filterStatus` and `filterCategory` states, ensuring the table updates instantly when a filter is changed.
    - **Rendering:** The component renders the `Pie` and `Bar` charts, the filter dropdowns, and a full HTML `<table>` of all feedback, mapping over the `filteredFeedback` list.
- **`AdminDashboard.css` (New File):** A dedicated CSS file was created to give the dashboard a professional, grid-based layout. It includes styles for the charts, the filter controls, and the feedback table, with color-coded tags for feedback status (`.status-open`, `.status-resolved`, etc.).