Student Registration Angular Application – Architecture and Technologies

1. Overview

This project is a **Student Registration and Course Statistics** application built with **Angular 16**. The system allows users to register students, assign them to courses, and visualize course registration statistics using charts. The project follows a **component-based architecture**, adhering to Angular best practices.

---

2. Architecture & Folder Structure

Components

- **student-form**: Handles the student registration form. Uses **Reactive Forms** for input validation, including custom validators for courses selection.
- **course-statistics**: Displays course statistics using **ng-apexcharts**. Shows courses vs. registered student percentages in a **pie chart**.
- **navigation-taps**: Navigation bar for switching between the Student form and Statistics pages.
- **shared/error-modal**: Handles global error messages with a **Bootstrap modal** using **@ng-bootstrap/ng-bootstrap**.

Services

- **courses.service.ts**: Provides course data from the backend (JSON file or mock service).
- **student.service.ts**: Manages student registration data.
- **shared/error-modal.service.ts**: Centralized service to trigger error modals from any component.

Assets

- **db.json**: Mock backend file storing courses and student data.

Routing

- **app-routing.module.ts**: Defines application routes:
  - `/Students` → Student registration form
  - `/Statistics` → Courses statistics chart

Modules

- **app.module.ts**: Imports Angular modules, Material components, ReactiveFormsModule, HttpClientModule, and ng-bootstrap modules.

---

3. Key Technologies & Libraries

- **Angular 16** – Frontend framework for building SPA.
- **Angular Material** – UI components such as **mat-select** for course selection.
- **Reactive Forms** – Form handling with built-in and custom validators.
- **ng-apexcharts** – Data visualization (pie chart for course statistics).
- **Bootstrap & ng-bootstrap** – Styling and modal dialogs.
- **TypeScript** – Strongly typed language for Angular development.
- **RxJS** – For handling asynchronous data with Observables.
- **JSON** – Mock backend storage.

---

4. Design Highlights

- **Component-based structure** allows clear separation of UI and logic.
- **Reactive Forms** provide robust validation and accessibility.
- **Custom Validators**: For example, `courseLimitValidator` enforces minimum and maximum course selection.
- **Error Handling**: Centralized modal-based error messages using `ErrorModalService` and `ErrorModalComponent`.
- **Accessibility**: Use of `aria-label`, `aria-invalid`, and `role="alert"` for form fields and navigation.
- **Charts**: Pie chart visualizations show course enrollment percentage, calculated by dividing the number of students per course by the total students.

---

5. Observations & Best Practices

- The app uses **services for data access** to decouple components from backend logic.
- All asynchronous operations (like `getStudents()` and `getAllCourses()`) use **Observables** for easy subscription and error handling.
- **Error modals** provide a user-friendly way to handle failures without relying on `alert()`.
- The application is modular and scalable, allowing the addition of new components and services with minimal impact on existing code.

---

6. Development Server

- Run **ng serve** for a dev server. Navigate to **http://localhost:4200/**. The application will automatically reload if you change any of the source files.
- Open **\src\assets\Data** in terminal and run **json-server --watch db.json** to make db.json act as a fack server