TheKiranAcademy

SamsTrack Rest API Documentation

About the Project

SamsTrack

Is a Spring Boot REST API project designed to streamline the process of managing student attendance in educational institutions. It allows faculty members to record attendance for specific subjects and provides interfaces for viewing and managing these records. The primary entities in the system are Students, Subjects, Faculty, and Attendance Records. The application aims to reduce paperwork, improve accuracy, and simplify attendance tracking.

Objective:

- ✓ Automate attendance tracking through a REST API to streamline integration with existing systems.
- ✓ Ensure data accuracy with automated validation and storage of attendance records.
- ✓ Facilitate seamless integration with educational platforms to enhance faculty and administrative workflows.

Scope of the Project:

The scope of the SamsTrack project involves developing a REST API to manage student attendance, including functionalities for handling Students, Subjects, Faculty, and Attendance Records.

Project Structure

Overview of Project Architecture: The project follows a layered architecture pattern, with distinct layers for handling different aspects of functionality and data management.

Layers Description:

1. Controller Layer: Responsible for handling incoming HTTP requests, processing them, and returning appropriate responses. Controllers act as the entry point to the application and delegate business logic to the service layer.

2. Service Layer: Acts as an intermediary between the controller and the data access layer (DAO). It contains business logic and orchestrates interactions between different parts of the application. Services handle tasks such as validation, data manipulation, and transaction management.

3. DAO (Data Access Object) Layer: Responsible for interacting with the database and performing CRUD (Create, Read, Update, Delete) operations on the underlying data. DAOs

abstract away the details of database interactions and provide a clean interface for the service layer to work with.

Responsibilities of each layer:

- **Controller Layer:**
  - Handle incoming HTTP requests.
  - Validate request parameters and payload.
  - Invoke appropriate methods in the service layer.
  - Format and return HTTP responses.
- **Service Layer:**
  - Implement business logic and rules.
  - Coordinate interactions between controllers and DAOs.
  - Perform data manipulation and transformation.
  - Manage transactions and ensure data integrity.
- **DAO Layer:**
  - Provide CRUD operations for accessing and modifying data.
  - Translate database queries and commands into appropriate SQL
  - statements.
  - Handle database transactions and connections.
  - Abstract away database-specific details from the service layer.

Used Technology Stack

- ✓ Backend Framework: Spring Boot 2.5.6
- ✓ Database: MySQL 8
- ✓ Database Framework: Hibernate 5.6
- ✓ Java: JDK 1.8
- ✓ Build Tool: Maven
- ✓ Development Environment: Eclipse
- ✓ API Testing Tool: Postman

Modules in Project

1. Student Module
2. Subject Module
3. Faculty Module
4. Attendance Module
5. User Module

In Detail About Modules and Its APIs

Student Module:

Description:

This module manages operations related to students, such as creating student, updating student, retrieving student, and deleting student records etc...

API Endpoints:

GET `/student/ get-all-students`:

Fetches a list of all students in the database, providing an overview of all registered students. Users can view all student records.

POST `/student/ add-student`:

Allows the creation of a new student record by providing details such as name and email. Users can add new students.

GET `/student/get-student-by-id/{id}`:

Retrieves details of a specific student identified by their ID, enabling users to view individual student information. Users can access specific student details.

PUT `/student/update-student:`

Updates the details of an existing student, allowing users to modify student information. Users can update student records.

DELETE `/student/delete-student/{id`

Deletes a specific student from the database, enabling users to remove student records. Users can delete student records.

Subject Module

Description:

This module manages all operations related to subjects within the system. It allows users to create new subjects, update existing subject information, retrieve details of all subjects or specific subjects, and delete subjects from the database. This functionality ensures that the subject records are accurately maintained and up-to-date.

API Endpoints:

GET `/subject//get-all-subjects`

Fetches a list of all subjects, providing an overview of all available subjects in the database. Users can view the complete subject catalog.

POST `/subject/add-subject`

Allows the creation of a new subject record by providing details such as the subject name. Users can add new subjects to the system, ensuring the curriculum is up-to-date.

GET `/subject/ get-subject-by-id/{id}`

Retrieves the details of a specific subject identified by its ID, enabling users to view comprehensive information about a particular subject. Users can access and review specific subject details.

PUT `/subject/ update-subject`

Updates the details of an existing subject, allowing users to modify subject information as needed. Users can update subject records to reflect changes in the curriculum.

DELETE `/subject/delete-subject/{id}`

Deletes a specific subject from the database, enabling users to remove outdated or unnecessary subject records. Users can manage and clean up the subject catalog.

Faculty Module

Description:

This module is responsible for managing all operations related to faculty members within the system. It enables users to create new faculty records, update existing faculty information, retrieve details of all faculty or specific faculty members, and delete faculty records from the database. This functionality ensures that the faculty information is accurate, current, and easily accessible, supporting the overall management of educational resources and personnel.

API Endpoints:

GET `/faculty/get-all-faculties`

Fetches a list of all faculty members, providing an overview of all registered faculty in the database. Users can view the complete list of faculty members.

POST `/faculty/add-faculty`

Allows the creation of a new faculty record by providing details such as name and email. Users can add new faculty members to the system, ensuring that the faculty roster is comprehensive.

GET `/faculty/get-faculty-by-id/{id}`

Retrieves the details of a specific faculty member identified by their ID, enabling users to view individual faculty information. Users can access and review detailed information about specific faculty members.

PUT `/faculty/update-faculty`

Updates the details of an existing faculty member, allowing users to modify faculty information as needed. Users can update faculty records to reflect changes in personnel.

DELETE `/faculty/delete-faculty/{id}`

Deletes a specific faculty member from the database, enabling users to remove outdated or unnecessary faculty records. Users can manage and clean up the faculty roster.

Attendance Module

Description:

This module helps record and manage student attendance data. Faculty members can create and track attendance records for their subjects and students.

API Endpoints:

GET `/attendance/get-all-attendance-records`

This endpoint lets you retrieve a list of all recorded attendance data. It's useful for checking past attendance and generating reports.

POST `/attendance/add-attendance`

This endpoint allows faculty to create a new attendance entry. When creating a record, the faculty member provides the subject ID, a list of IDs for students who attended, and the date of the attendance and attendance id system will automatically for the attendance record also store the faculty member's ID as the attendance taker.

User Module

Description:

The User module in the SamsTrack application is designed to handle user-related functionalities, managing roles and authentication to ensure secure access to the system. Here’s a detailed explanation of its components and functionalities

API Endpoints:

- ✓ **POST** `/user/login-user`

  Purpose: Authenticates a user based on the provided credentials (username and password) and returns user details.

Request: Accepts a JSON body containing user login details, such as username and password.

Response: Returns user information if authentication is successful, null if authentication fails.

✓ **POST** `/user/register-user`

Purpose: Registers a new user in the system by creating a new user record with the provided details.

Request: Accepts a JSON body containing user information such as username, email, and password.

Response: Returns the newly registered user's details or an error message if the registration fails.

✓ **GET** `get-user-by-username/{username}`

Purpose: Fetches and returns the user details associated with the specified username.

Request: Accepts a path variable username to identify the user.

Response: Returns the user details or an error message if the user is not found.

✓ **GET** `/get-all-user`

Purpose: Retrieves and returns a list of all registered users.

Request: Does not require any parameters or body

Response: Returns a list of users or an empty list if no users are found.

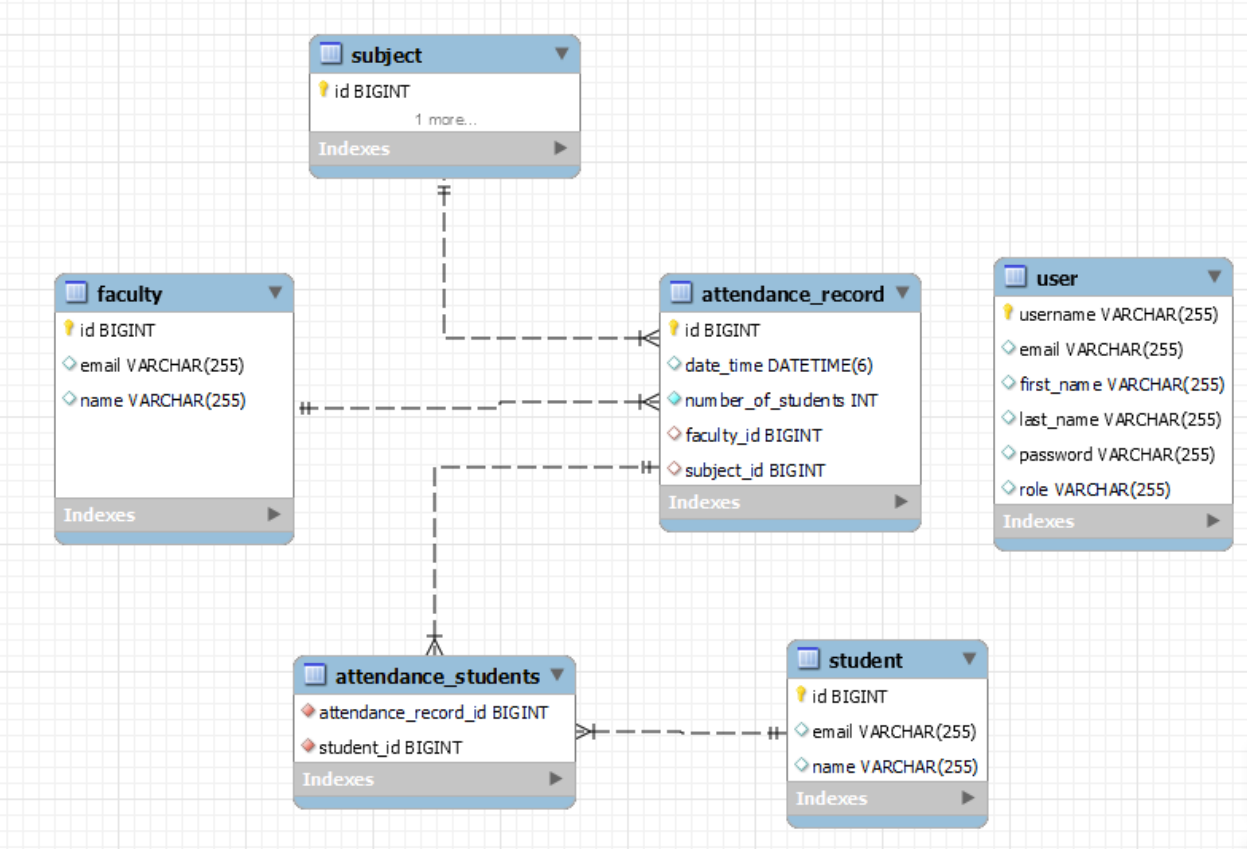
✓ **DELETE** `/get-all-user`

- **Purpose**: Deletes the user with the specified username from the system.
- **Request**: Accepts a request parameter `username` to identify the user to be deleted.
- **Response**: Returns a confirmation message indicating whether the user was successfully deleted or an error message if the deletion fails.

Database

Click Here to Download DB

Entity-Relationship (ER) diagram for the SamsTrack application



Sample Interview Questions

1. What is Spring Boot, and why did you choose it for this project?

Spring Boot is a framework that simplifies the development of Java-based applications by providing default configurations and a wide range of built-in

functionalities. It was chosen for its ease of use, rapid development capabilities, and extensive community support.

2. Can you explain how JPA and Hibernate are used in this project?

JPA (Java Persistence API) is a specification for accessing, persisting, and managing data between Java objects and a relational database. Hibernate is the JPA implementation used in this project. It helps in mapping Java classes to database tables and provides an easy way to perform database operations.

3. How does the application handle multiple students in an attendance record?

The application uses a many-to-many relationship between `AttendanceRecord` and `Student` entities. When creating an attendance record, a list of student IDs is provided, which are then associated with the attendance record in the database.

4. What are the benefits of using MySQL as the database for this project?

MySQL is a widely used open-source relational database management system. It is known for its reliability, robustness, and ease of use. Additionally, it integrates well with Spring Boot and provides efficient data storage and retrieval capabilities.