My project name is **Member Enrollment API**

This is a Spring Boot web application that provides endpoints for member enrollment.

**Setup**

1. Install Java Development Kit (JDK) version 17 or later.

2. Install IntelliJ for building the project.

* Use the language Java and the built system is Maven.

3. Clone this repository to your local machine GIT.

4. Install Postman

**Build and Run**

To build and run the application, open the project in IntelliJ and run the `MemberEnrollmentApiApplication` class.

**API Endpoints**

- GET `/api/members/{id}`: Get member details by ID.

- POST `/api/members`: Create a new member enrollment.

- PUT `/api/members/{id}`: Update member details by ID.

- DELETE `/api/members/{id}`: Delete member details by ID.

**Database**

The application uses an in-memory H2 database for storing member details.

I have created two members’ Enrollment initially in that database.

**Testing**

Unit tests are provided for the controller and repository classes. You can run the tests using the `MemberControllerTest` class.

**Verify that new member enrollments are successfully created and stored in the local database**

1. On the Web browser type <http://localhost:8080/h2-console> then it will take you to the login page.

2. Then leave the default settings (JDBC URL should be jdbc:h2:mem:testdb) and click "Connect."

3. Then it will take you to H2 Database Console.

4. There you can use SQL queries to know the present and created member enrollment in that.

**Postman**

1. You can do operations GET, POST, DELETE, PUT operations over here

2. The URL for GET: <http://localhost:8080/api/members/1>

POST URL: <http://localhost:8080/api/members>.

**Here is the complete code and path of how I set dependency and connected to h2 database**

Step 1:

I have created a project name MemberEnrollmentAPI

step 2:

I have pom.xml in the MemberEnrollmentAPI project where i have added added dependency and a plugin.

<Deoendencies>

Spring Boot Starter Web, Spring Boot Starter Data JPA, H2 Database, JAXB API, Jackson Databind , JUnit Jupiter API, Spring Boot Starter Test.

<plugins>

Maven Compiler Plugin, Spring Boot Maven Plugin

Step 3:i have created the application.properties in the path “src/main/resources”

The application.properties file contains the configuration for the H2 in-memory database. The URL specifies the database connection and creates an in-memory database named "testdb." The username is set to "sa" with an empty password. The driver class is specified as "org.h2.Driver," and the Hibernate dialect is set to H2. The property "spring.jpa.hibernate.ddl-auto" is set to "create," which automatically generates the database schema on application startup. Additionally, "spring.h2.console.enabled" is set to "true," enabling access to the H2 database console for easy management and monitoring during development.

Step 4: i have created main in MemberEnrollmentApiApplication class the path is: MemberEnrollmentAPI/src/main/java/com/yourcompany/memberenrollmentapi/MemberEnrollmentApiApplication.java

The "MemberEnrollmentApiApplication" class serves as the main entry point for the Spring Boot application. It is annotated with "@SpringBootApplication," which combines the annotations "@Configuration," "@EnableAutoConfiguration," and "@ComponentScan." This enables automatic configuration and component scanning, making it a fully functional Spring Boot application. The "main" method is crucial, responsible for launching the application. When executed, the application initializes and sets up the necessary components, allowing it to handle incoming requests and manage member enrollment operations. The class plays a pivotal role in starting and running the Member Enrollment API application smoothly.

Step 5:i have created Member class and the path is

MemberEnrollmentAPI/src/main/java/com/yourcompany/memberenrollmentapi/entity/Member.java

The "Member" class is an essential entity representing the member details in the application. It uses the "@Entity" annotation, making it a JPA entity, enabling seamless integration with the database. The class contains attributes like "firstName," "lastName," "email," and "birthdate" to store member-specific data. Two constructors are available: a default one and a parameterized one to initialize member objects. Getter and setter methods are implemented to access and modify member properties. Additionally, "equals()" and "hashCode()" methods are provided for proper comparison in collections. This class facilitates the mapping of member data, retrieval, and manipulation within the application.

Step 6: I have created Member Repository interface and the path is

MemberEnrollmentAPI/src/main/java/com/yourcompany/memberenrollmentapi/repository/MemberRepository.java

The "MemberRepository" interface is a Spring Data JPA repository that extends the "JpaRepository" interface. By extending "JpaRepository<Member, Long>," it inherits powerful database CRUD (Create, Read, Update, Delete) operations for the "Member" entity class. Spring Data JPA automatically generates implementation code for these operations, making it easy to interact with the database. The "@Repository" annotation indicates that this interface is a Spring-managed repository, allowing it to be automatically detected and utilized by the Spring application context. This interface acts as an intermediary between the application and the underlying database, providing a convenient and standardized way to perform database operations related to the "Member" entity.

Step 7: I have created MemberController class and the path is

MemberEnrollmentAPI/src/main/java/com/yourcompany/memberenrollmentapi/controller/MemberController.java

The "MemberController" class is a Spring REST controller responsible for handling HTTP requests related to member enrollment. It utilizes the "MemberRepository" to interact with the database and perform CRUD operations on "Member" entities. The class defines two endpoints:

1.GET endpoint ("/api/members/{id}") to retrieve a member by their ID.

2.POST endpoint ("/api/members") to create a new member.

The "CommandLineRunner" interface is implemented to execute the "initializeMembers()" method during application startup, populating the database with sample data if it's empty. The class uses annotations like "@RestController" for RESTful functionality, "@GetMapping" and "@PostMapping" for mapping endpoints, and "@Autowired" for dependency injection. The methods handle HTTP requests, return appropriate responses, and throw a "ResponseStatusException" if a member is not found.

Step 8: I have created MemberControllerTest and the path is

MemberEnrollmentAPI/src/test/java/com/yourcompany/memberenrollmentapi/entity/MemberControllerTest.java

The "MemberControllerTest" class is a JUnit test class that performs integration testing for the "MemberController" class. It uses the Spring Boot testing framework and "TestRestTemplate" to send HTTP requests to the API endpoints exposed by the controller. The test verifies the functionality of the "getMemberById" method by creating a sample member, saving it to the database, and then making a GET request to retrieve the member's details. The response is then checked for correctness, including the status code, member properties, and data consistency. This class allows developers to ensure that the API endpoints work as expected and handle various scenarios appropriately during the testing process