Documentation for MessageRelay System

Introduction:

The Message Relay system is a simple RESTful API that allows users to send and retrieve messages. The API is built using **Spring Boot 3.2.2** and utilizes an in-memory H2 database for message persistence. It features basic functionalities to send a message and retrieve messages for a specific receiver.

MessageRelay

Technologies and Tools Used:

- **Spring Boot 3.2.2:** Main framework for building the application.
- **Spring Data JPA:** For database interaction and CRUD operations.
- **Spring Boot Starter Security:** For security features.
- **Jakarta Persistence API:** For JPA functionality.
- **H2 Database:** Used for in-memory database testing.
- **JUnit 5:** For unit testing.

Detailed Explanation of the Project:

1. Entry Point: `MessageRelayApplication.java`

This is the entry point of the Spring Boot application. It initializes the Spring Boot context and launches the application.

2. Model: `Message.java`

The `Message` class represents the message entity and uses JPA annotations to map the entity to the `Message` table.

- **Key Fields**:
  - `content`: The body of the message.
  - `sender`: The person who sends the message.
  - `receiver`: The intended recipient of the message.
- **Annotations**:
  - `@Entity`: Specifies that this class is a JPA entity.
  - `@Id`: Defines the primary key.

- `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Sets up auto-increment for the `id` field.

3. Persistence Layer: `MessageRepository.java`

`MessageRepository` extends **JpaRepository**, providing methods to perform database operations on `Message` objects without having to write SQL queries.

4. Service Layer: `MessageService.java`

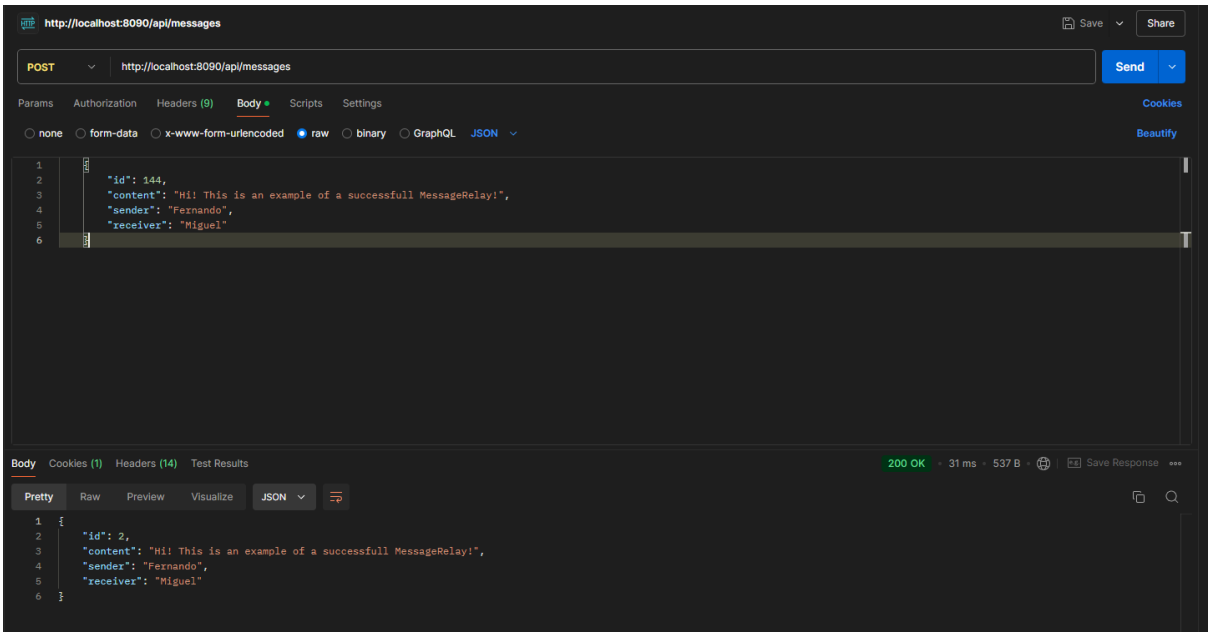
The service layer encapsulates the business logic. It interacts with `MessageRepository` to send and retrieve messages.

- **Key Methods**:
  - `sendMessage(Message message)`: Saves the message to the database.
  - `getMessagesForReceiver(String receiver)`: Retrieves all messages for a specific receiver using `stream()` and `filter()`.

5. Controller Layer: `MessageController.java`

The controller handles HTTP requests for sending and retrieving messages.

- **Endpoints**:
  - `POST /api/messages`: Sends a new message.
  - `GET /api/messages/{receiver}`: Retrieves all messages for the specified receiver.



Database Configuration: `application.properties`

The project uses the H2 in-memory database, which is configured as follows:

properties

```
server.port=8090
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

The H2 database is ephemeral, meaning all data will be lost after the application stops. This setup is useful for testing and development purposes.