**RECIPE REST API**

RECEPI REST SERVICE API:

* CRUD Operation (GET/POST/PUT/DELETE)
* Swagger UI
* Junit
* Source code availabile at GIT (<https://github.com/port2pk/RecipeRESTAPI>.)
* My SQL Database integration.
* Logging
* Spring Security with JWT web authentication.
* Centralized Exception Handling.

**DETAILS:**

It is just a sample demo for demonstrate the rest service. The below given feature has been added.

1. A sample CRUD operation for recipe add/ update / delete / retrieve.

**GET Method:**

URL: <http://localhost:8080/users/recipes>

Status: 200 OK

Header : Authorization – Bearer “JWT token”

Output:

[

{

"recipeId": 1,

"recipeCd": "Coconut Latte Overnight Oats",

"recipeCreationDt": "2021-05-18T18:30:00.000+00:00",

"recipeModificationDt": "2021-05-18T18:30:00.000+00:00",

"noOfPerson": 3,

"ingredientsId": 1,

"active": true

},

{

"recipeId": 2,

"recipeCd": "Coconut Latte Overnight Oats",

"recipeCreationDt": "2021-05-18T18:30:00.000+00:00",

"recipeModificationDt": "2021-05-18T18:30:00.000+00:00",

"noOfPerson": 4,

"ingredientsId": 2,

"active": false

}

]

**POST Method:**

URL : <http://localhost:8080/users/recipe>

Status: 201 Created

Header: Authorization – Bearer “JWT token”

Input: {

        "recipeId": **null**,

        "recipeCd": "Coconut Latte Overnight Oats",

        "recipeCreationDt": "2021-05-18T18:30:00.000+00:00",

        "recipeModificationDt": "2021-05-18T18:30:00.000+00:00",

        "noOfPerson": 4,

        "ingredients": {

            "ingredientsId": **null**,

            "items": "Rolled Oats, Lite Coconut Milk, Brewed Coffee, Maple Syrup, Ground Cinamon"

        }

    }

**PUT Method:**

URL : http://localhost:8080/users/recipe/4

Status: 200 OK

Header: Authorization – Bearer “JWT token”

Input: {

        "recipeId": 4,

        "recipeCd": "Coconut Latte Overnight Oats",

        "recipeCreationDt": "2021-05-18T18:30:00.000+00:00",

        "recipeModificationDt": "2021-05-18T18:30:00.000+00:00",

        "noOfPerson": 5,

        "ingredients": {

            "ingredientsId": 4,

            "items": "Rolled Oats, Lite Coconut Milk, Brewed Coffee, Maple Syrup, Ground Cinamon"

        }

    }

Output: {

    "recipeId": 4,

    "recipeCd": "Coconut Latte Overnight Oats",

    "recipeCreationDt": "2021-05-18T18:30:00.000+00:00",

    "recipeModificationDt": "2021-05-18T18:30:00.000+00:00",

    "noOfPerson": 5,

    "ingredientsId": 4,

    "active": **false**

}

**DELETE Method:**

URL : http://localhost:8080/users/recipe/4

Status : 204 NO Content

Header: Authorization – Bearer “JWT token”

1. **Spring Security with JWT web authentication:**

**JWT (JSON Web Token) :** For rest api security , jwt token required.

To obtain the token need to execute this URL.

For getting JWT Token:

URL: <http://localhost:8080/authenticate>

Input: {

    "username":"user",

    "password":"password"

}

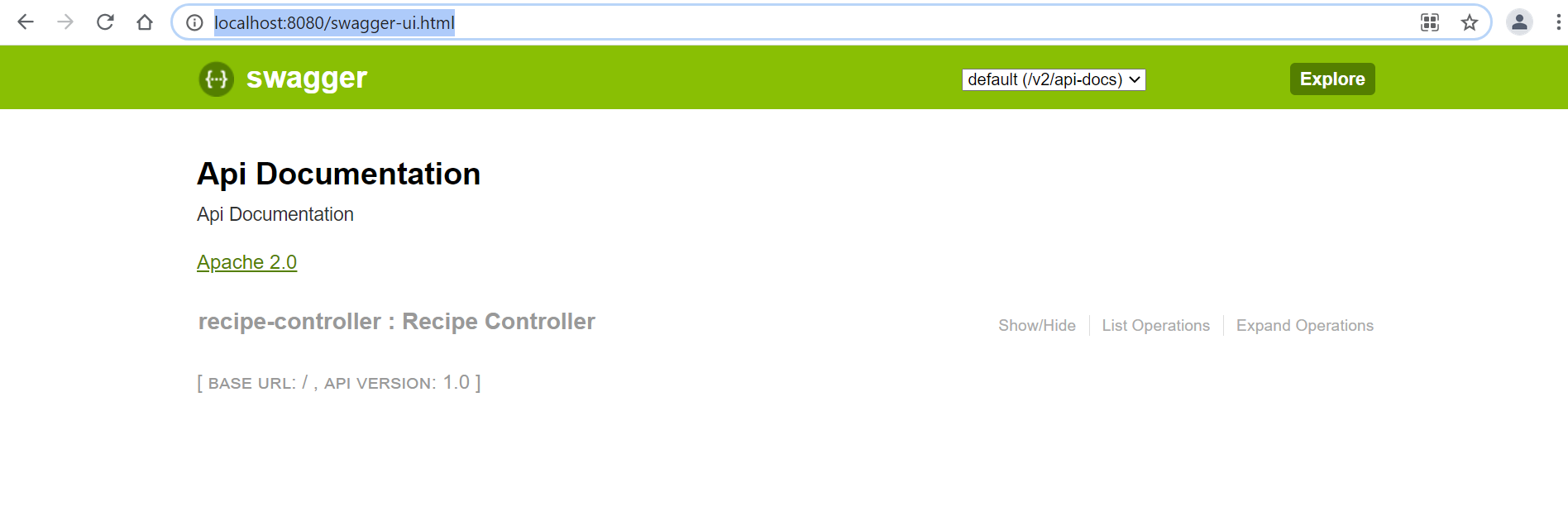
Output: {

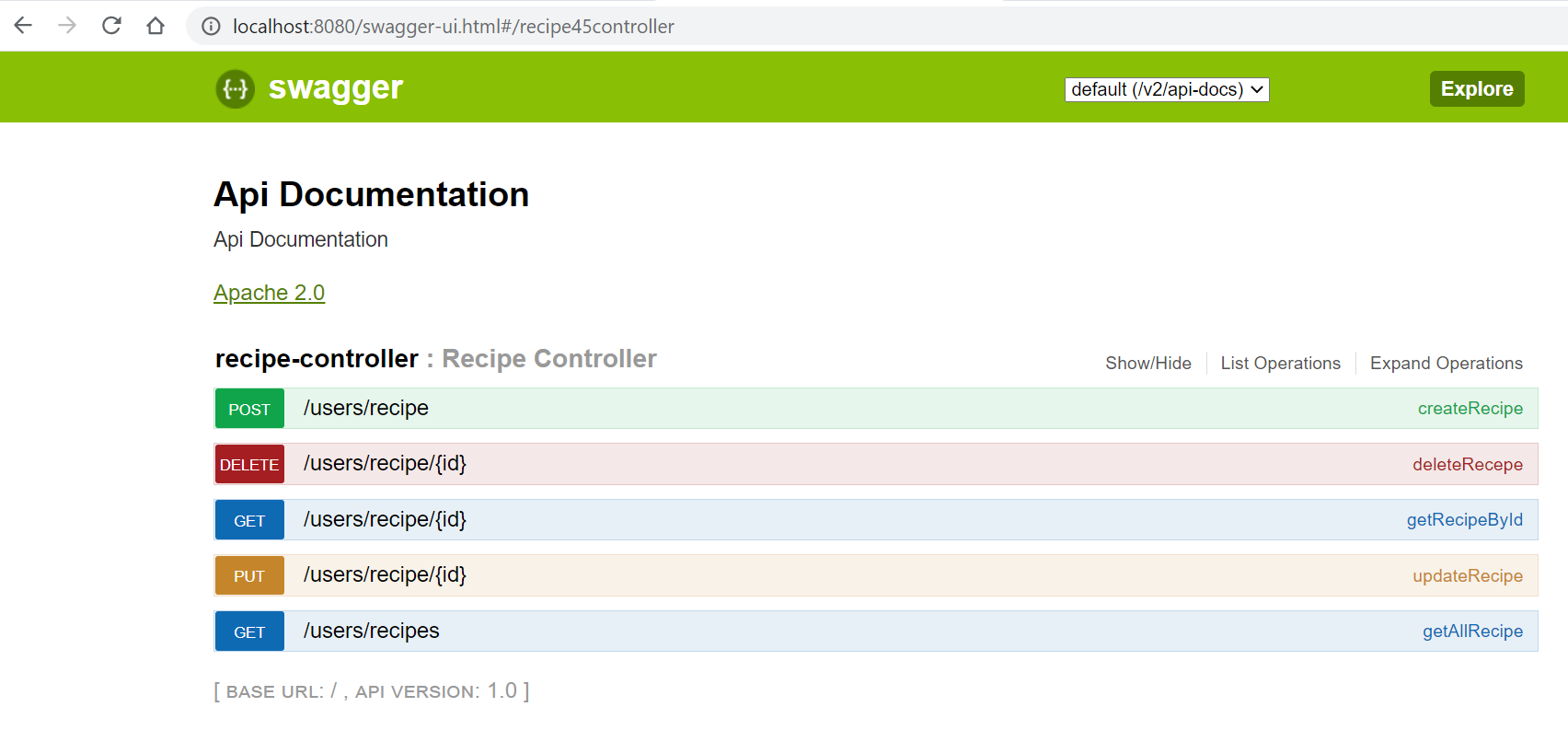
    "jwt": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiZXhwIjoxNjIxNDUxNzI5LCJpYXQiOjE2MjE0MTU3Mjl9.N6J0vEVA1fPovXPw9MyMfFEznpST1K7jyzK7EvmILco"

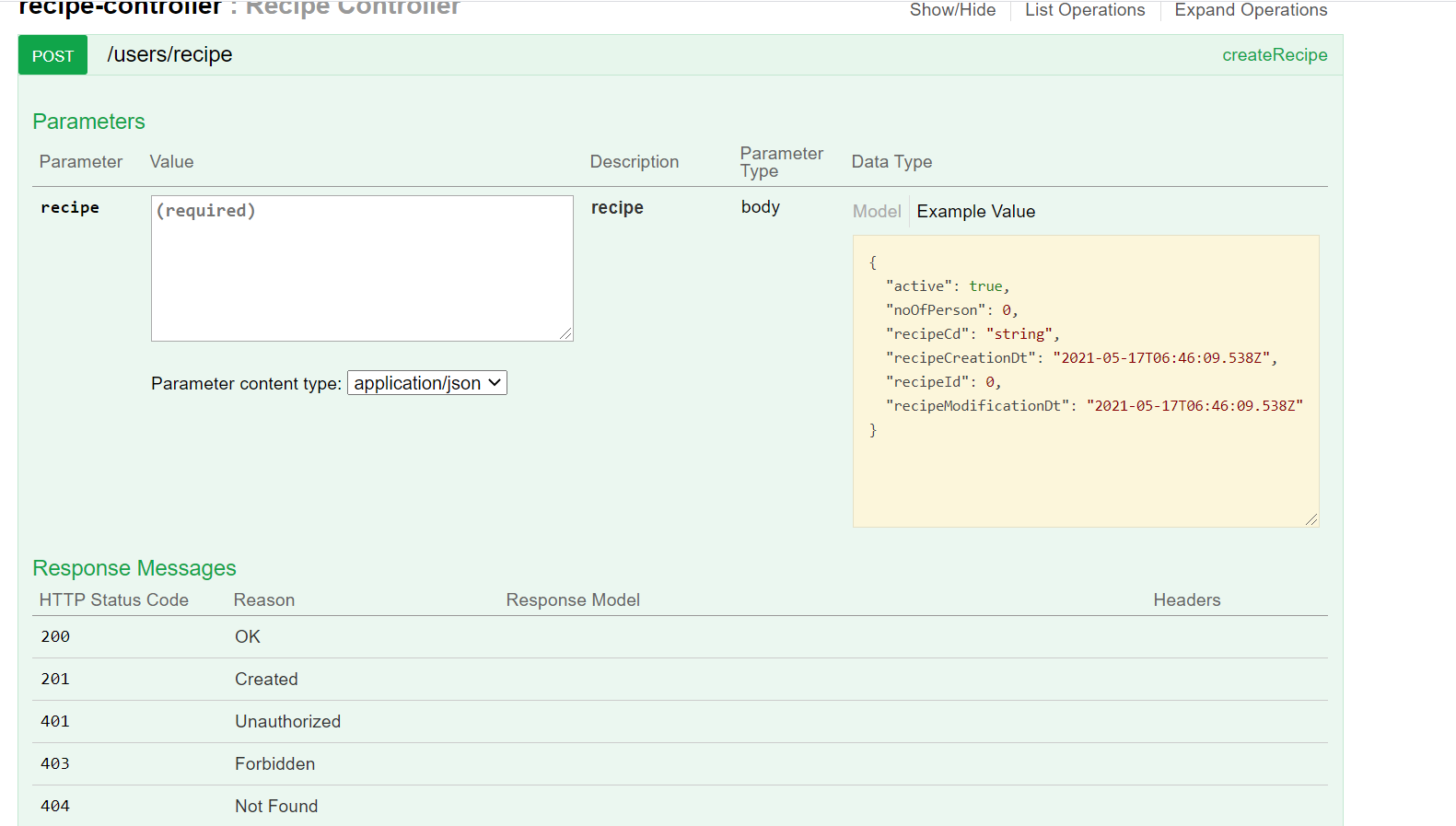
}

1. Integration of Swagger UI:

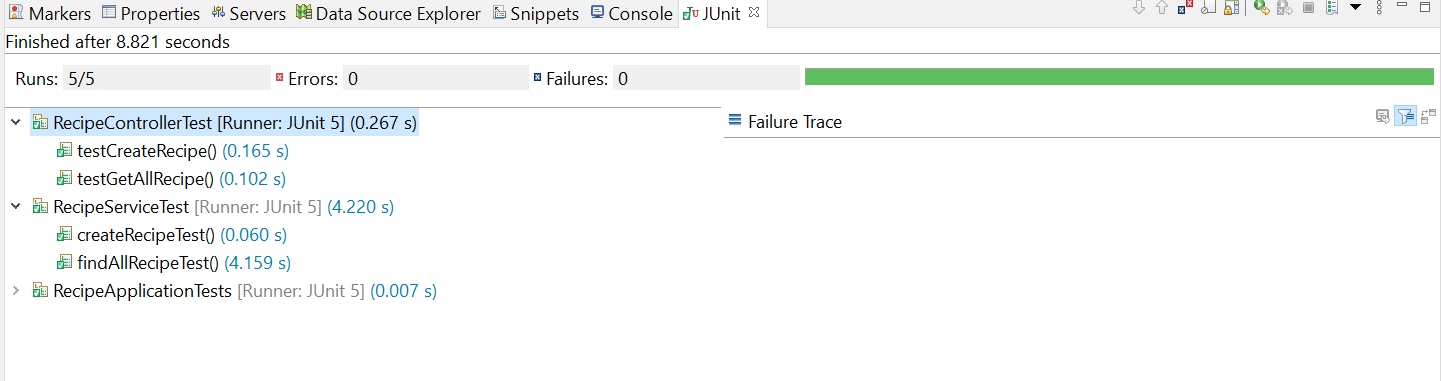
URL: <http://localhost:8080/swagger-ui.html>







1. Junit Test integeration



As per the testing framework integration, Junit Test sample cases has been added for demo purpose. For below given layers

Controller

Service

It can be added for other layers as well.

1. **Centralized Exception Handler**:

There is centralized exception handling for custom exception.

@ControllerAdvice

**public** **class** ExceptionHelper **extends** ResponseEntityExceptionHandler {

**private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(ExceptionHelper.**class**);

@ExceptionHandler(value = { ExpiredJwtException.**class** })

**public** ResponseEntity<Object> handleExpiredJwtException(ExpiredJwtException ex, WebRequest req) {

***log***.error("ExpiredJwtException: ", ex.getMessage());

ExceptionResponse exceptionResponse = **new** ExceptionResponse(**new** Date(),ex.getMessage(),req.getDescription(**false**));

**return** **new** ResponseEntity<Object>(exceptionResponse, HttpStatus.***UNAUTHORIZED***);

}

@ExceptionHandler(value = { SignatureClaimException.**class** })

**public** ResponseEntity<Object> handleSignatureClaimException(SignatureClaimException ex, WebRequest req) {

***log***.error("SignatureClaimException: ");

ExceptionResponse exceptionResponse = **new** ExceptionResponse(**new** Date(),ex.getMsg(),req.getDescription(**false**));

**return** **new** ResponseEntity<Object>(exceptionResponse, HttpStatus.***UNAUTHORIZED***);

}

@ExceptionHandler(value = { NoEntityFoundException.**class** })

**public** ResponseEntity<Object> handleNoEntityFoundException(NoEntityFoundException ex, WebRequest req) {

***log***.error("Exception: ", ex.getMessage());

ExceptionResponse exceptionResponse = **new** ExceptionResponse(**new** Date(),ex.getMsg(),req.getDescription(**false**));

**return** **new** ResponseEntity<Object>(exceptionResponse, HttpStatus.***NOT\_FOUND***);

}

@ExceptionHandler(value = { Exception.**class** })

**public** ResponseEntity<Object> handleAllException(Exception ex, WebRequest req) {

***log***.error("Exception: ", ex.getMessage());

ExceptionResponse exceptionResponse = **new** ExceptionResponse(**new** Date(),ex.getMessage(),req.getDescription(**false**));

**return** **new** ResponseEntity<Object>(exceptionResponse, HttpStatus.***INTERNAL\_SERVER\_ERROR***);

}

}

Note: Apart from that traditional try, catch block has been used to show exception handling in traditional way of error propagation.

1. All the source code available at <https://github.com/port2pk/RecipeRESTAPI>.
2. **Logging** :

Spring Boot provides several in built feature of logging, sample demo has been given.

1. Database : Just a sample data set has been provided.

MySql:

spring.datasource.driver=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/recipeDb

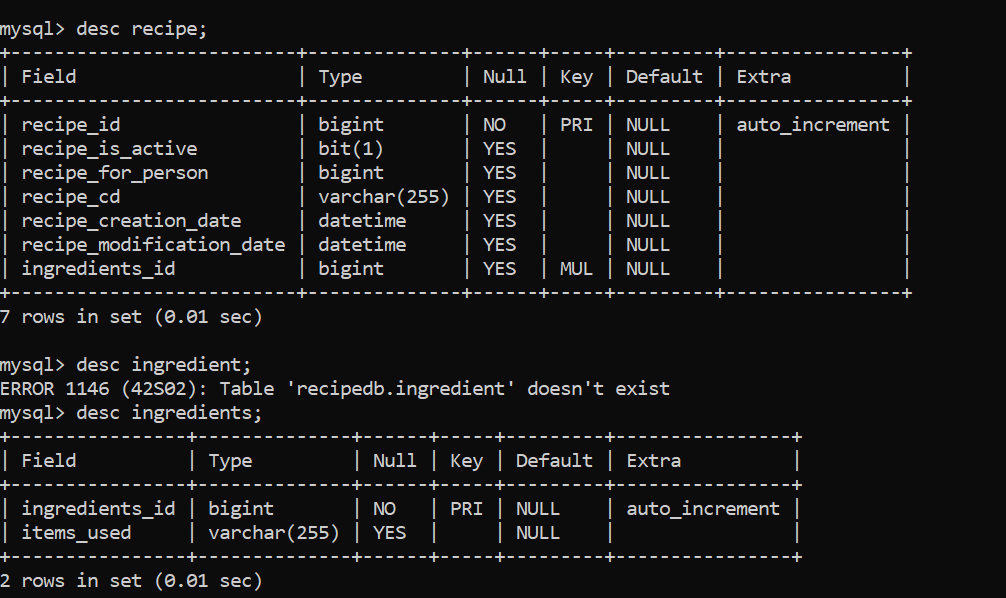
spring.datasource.username=root

[spring.datasource.password=miku@2210](mailto:spring.datasource.password=miku@2210)

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.database-platform = org.hibernate.dialect.MySQL5Dialect



1. Logging: it is one of the most widely used feature being used in application for debugging purpose.

Currently in this application a general logger using as a sample.

Future Scope (TO-DO LIST)

Currently this project has been developed for a small level demos of rest service API, with limited facility but there are still more chances to improvise this.

This project can be improvized by adding some more features:

1. Spring Security integration for rest service. Currently it is not available due to time unavailability.

JWT (JSON Web Token) security (),

OAuth2 Security.

1. Few more types of junit test case can be added like DAO layer, integration testing etc.
2. UI support can be added by using Angular as a front end.
3. At database end, ORM feature can be added in more effective ways by adding more complex scenarios.
4. Logging & Debugging.
5. Centralized Exception Handling.
6. Data validation.