Spring-RestAPI-Oracle VM Application

Abishek Arumugam Thiruselvi
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
abishekarumugam@icloud.com

***Abstract*— To implement the service design of OpenAPI and to deploy the client/server application in Oracle Cloud VM.**

Keywords—Oracle Cloud VM, Spring Framework and JUnit.

I. INTRODUCTION

A Client/Server application is developed using the spring framework. It's a Spring project with the dependencies "Spring Web". The Java application uses Java 11, maven 3.9.0, and Spring Boot 3.0.2.

The Spring application has a dedicated tomcat server used for testing the application locally. But we use Oracle Cloud VM for deploying the application. The domain name obtained from Cloud VM is used to test in swagger.io and Postman.

II. APPLICATION DESIGN

The application has source folders src/main/java, src/test/java and src/main/resources.

A. src/main/java

This source folder has 6 packages as shown in figure Fig. 2. 1. The details of the package are described below.

1. *Package: cu.assignment.controller*
   This package has the file "AudioController.java". It has the RestController which maps the RestAPI to the application.
2. *Package: cu.assignment.model*
   This package has the file "AudioItem.java". It is the class object which has all the details about an audio item.
3. *Package: cu. assignment.utils*
   This package has the file "DssAssignment1Application.java". It is the start of the spring boot application. Here we have used the @EnableAsync annotation to enable asynchronous processing in Spring. A bean named threadPoolTaskExecutor(bonus) is defined which is of type Executor and uses a ThreadPoolTaskExecutor to configure a thread pool.

B. src/test/java

1. *Package: cu. assignment.utils*
   This package has the file "DssAssignment1Application.java". It has test cases to test the client functionality by executing get and post concurrently.

C. src/main/resources

The source folder contains log files, reports and images for the readme file.

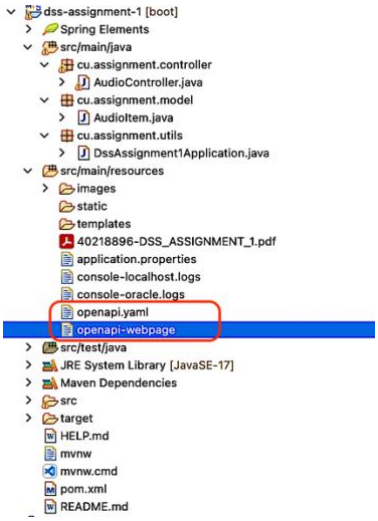

Fig. 2. 1. Project Structure

III. OPENAPI DESIGN

The class AudioItem has the properties like Artist Name, Track Title, Album Title, Track Number, Year, Number of Reviews, and The Number of Copies Sold. Get

- "/api/audio/artists/{id}" returns the audio item details of the specific id.
- "/api/audio/copies-sold" returns total number of copies sold.
- "/api/audio/artist/{name}" returns the list of audio items for a specific artist. Fig. 8.6.
- "/api/audio/artist/{name}/{property}" returns artists by name and a property.
- "/api/audio/all/artists" return all artist in JSON format.

A. Post

- "/api/audio/create" creates an audio item with the request in JSON format. Fig. 8.7.

B. OpenAPI on SwaggerHub

- OpenAPI test works only for cloud deployment. Does not work for localhost.
- for localhost OpenAPI test to work run the spring application and got to **http://localhost:8080/swagger-ui/index.html**
- The RestAPI is designed and tested through the swagger hub for testing and been included in the results subsection.
- Also, the postman was used to test the API functionality

C. Model Package

- In AudioItem.java, each property is a data member of the class. All the members have a getter and a setter. The class have a default superclass constructor.

IV. API IS HANDLED ON SERVER SIDE USING SERVLET

The AudioController has the @RestController annotations to access the Restful web services which are available in spring 4.0 and higher. It also has @RequestMapping which maps the web requests to the spring controller method.

A. Thread Safe Data Structure

The ExecutorService and CountDownLatch classes in java test cases can be used to ensure thread-safe operation in a multithreaded environment.

The ExecutorService manages the threads pool and executes the tasks in separate threads. The CountDownLatch waits for the threads to complete and blocks the incoming requests, and these threads are scheduled in a synchronization manner.

B. Multithreaded Operation

@Bean("threadPoolTaskExecutor") in java provides a thread pool that can be used to execute tasks in a multithreaded environment. It ensures that tasks are executed in a separate thread from the requestor but not the thread safety. The thread-safe is called in test cases.

V. CONCURRENT CLIENT TEST CASES

Enum Class "SpringBootTest.WebEnvironment" creates a web application context without defining any server by use of *RANDOM_PORT*.

TestRestTemplate is an alternative to RestTemplate, the clients in both cases are suitable for integration tests and can handle HTTP requests.

A. testAudioControllerGet

This method gets the response for calling all the artists in the database and maps all the artist to an array of strings and prints it in the console.

B. testAudioControllerPost

This method creates an audio item in the database and maps all the artist to an array of strings and prints it in the console.

C. testConcurrentClients

This method tests get and posts in clients concurrently on a given ratio. We use ExecutorService and CountDownLatch to ensure thread-safe operation.

D. Postman and SwaggerHub

This method tests get and posts in clients concurrently on a given ratio. We use ExecutorService and CountDownLatch to ensure thread-safe operation.

VI. CLOUD DEPLOYMENT

Oracle VM is used to deploy the spring project and the endpoints obtained are below.

A. Get Artists by ID :
http://155.248.235.48:8080/api/audio/artist/{id}

B. Get Copies Sold:
http://155.248.235.48:8080/api/audio/copies-sold

C. Get Artists by name:
http://155.248.235.48:8080/api/audio/artist/{name}

D. Create Audio Item:
http://155.248.235.48:8080/api/audio/create

E. Get Property value of Item by name:
http://155.248.235.48:8080/api/audio/artist/{name}/{property}"

F. Get all artist in JSON:
http://155.248.235.48:8080/api/ audio/all/artists

VII. INSTRUCTIONS TO RUN THE APPLICATION

Steps to install, run and test the application in cloud. Prerequisites are AWS login and npm installed on the local machine.

STEP 1.	Prerequisite: Git, Java17 or greater, maven 3.9 or higher.
STEP 2.	git clone https://github.com/abishekat/dss-assignment-1
STEP 3.	follow [3]
STEP 4.	cd dss-assignment-1
STEP 5.	mvn spring-boot:run
STEP 6.	Use Postman or OpenAPI (yaml: src/resources) to test the application.

Steps to install and run the spring application using install.

STEP 1.	Prerequisite: download and install Eclipse IDE plugin Spring Tools 3.
STEP 2.	Open the git perspective in eclipse and clone the git repository.
STEP 3.	Import the project into project explorer.
STEP 4.	run -> clean install
STEP 5.	run > java application / junit
STEP 6.	Terminal -> mvn spring-boot:run

VIII. RESULTS

Requirements satisfied are given below,

- Cloud Deployment – Spring Application.
- Get & Post of audio item with all properties.
- Section III.B - OpenAPI webpage and its public.
- Extended HttpServlet for AudioController.
- Thread safe: in-memory ds and countdownlatch.
- Servlet maintains a sum of the total number of copies sold.
- Servlet multithreaded with a 4-thread pool execution.
- 150 clients 5:1 get & post.
- GET: (1) returning a property value given an artist's name and (2) returning all the artists' data in JSON.
- POST: The POST response is a message confirming the audio item is stored in the database.
- Quality Attribute checking.

The test cases are run for 150 clients. The ratio of get and post is 5:1 and the graph show client vs time taken for the request to complete.

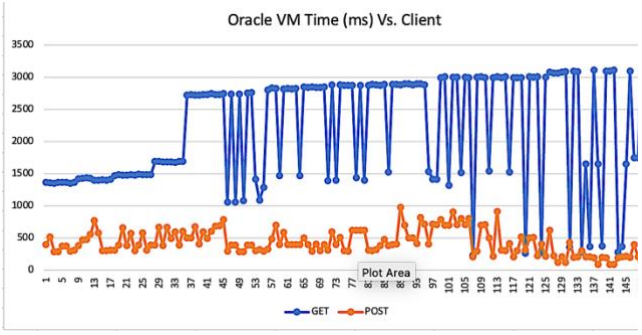

Fig. 8.1. Client Vs. Oracle VM server

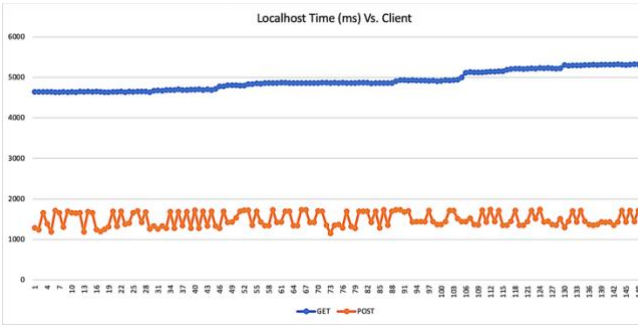

Fig. 8.2. Client Vs. localhost server

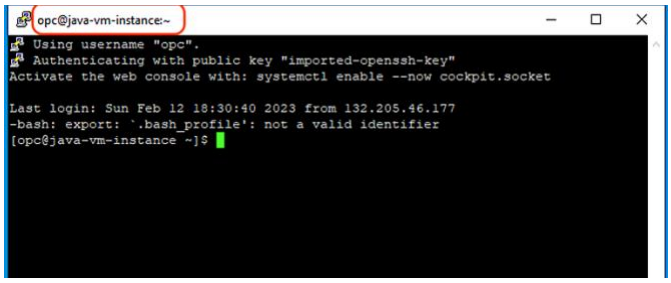

Fig. 8.3. SSH to Cloud VM

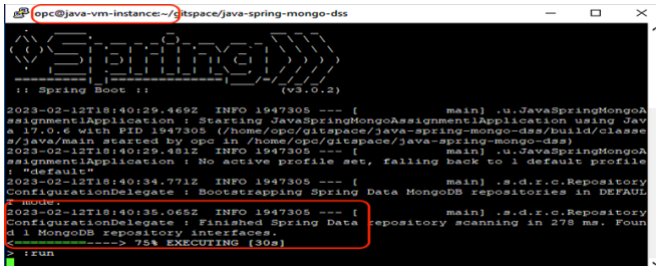

Fig. 8.4. Deploying and starting the web application

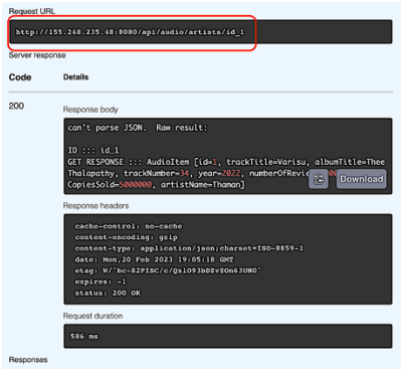

Fig. 8.5. Get: Artist by ID - OpenAPI

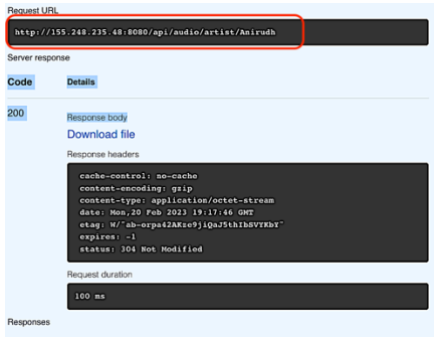

Fig. 8.6. Get: Artist by name – OpenAPI

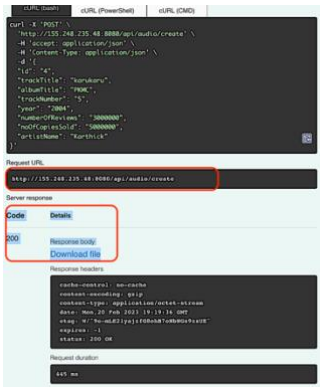

Fig. 8.7. Post: Create Audio Item - OpenAPI

Fig. 8.8. Get: Copies Sold before creating a new audio Item - OpenAPI

Fig. 8.9. Get: Copies Sold after creating a new audio Item - OpenAPI

REFERENCES

[1] https://moodle.concordia.ca/moodle/mod/lti/view.php?id=3433629

[2] https://medium.com/@contactkumaramit9139/spring-boot-integration-with-mongodb-c24e48f12ba7

[3] https://github.com/youyinnn/distributed_system_jetty_helloworld/blob/main/Oracle%20Cloud%20VM%20Setup.md