**Procedure to build Microservice application using spring boot and JAVA 8/11**

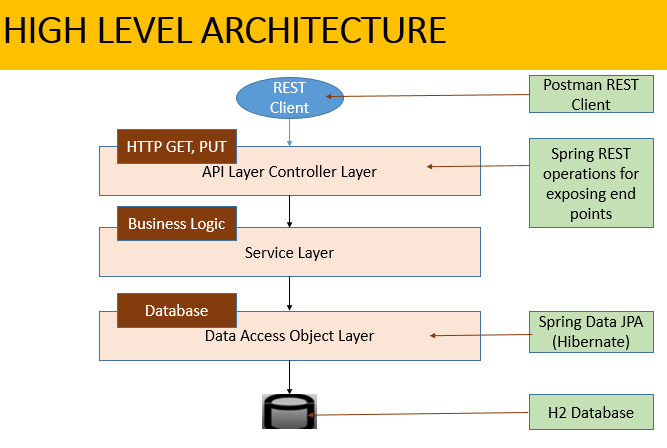
**Tools**

1) JDK 11

2) Maven 3.2+

3) IDE – STS or similar

4) Spring Boot – 2.3.4 RELEASE



|  |
| --- |
| **Requirement#1**  Create a simple micro service with 2 REST API’s to get and update user details using  In-memory database i.e h2  1) Get User Details (look at the example below to design the h2 database)  2) Update User Details (user can update any field     {                 "title": "mr",                 "firstn": "test",                 "lastname": "tsetlast",                 "gender": "male",                 "address": {                                "street": "12345 holling rd",                                "city": "Sydney",                                "state": "nsw",                                "postcode": 2000                 }     } |
| STEPS to implement Requirement#1  1. Created a spring boot project using “Spring Start Project” (with the following dependencies)  2. Provided the required details to configure the H2 database  3. Created a User bean which resembles the JSON data payload  4. Created a Repository interface extends JPARepository used for storing / fetching data from DB  5. Created a service class that makes a call to repository  6. Created a controller to expose the end-points |

|  |
| --- |
| **Requirement#2**  Insert 5 to 10 users in h2 db during the start-up to perform the above operations |
| Used CommandLineRunner interface to create a bean that is run when the application starts. We basically read in a list of users from users.json file and write them to database, whenever the application starts.  There are 6 user currently configured in the users.json file, present in the “./resources/json” folder |

|  |
| --- |
| **Requirement#3**  Implement the circuit breaker and transaction rollback for the database call. |
| Hystrix circuit breaker configuration is done using the below steps  1. Adding Hystrix starter and dashboard dependencies.  2. Adding @EnableCircuitBreaker annotation  3. Adding @EnableHystrixDashboard annotation  4. Adding annotation @HystrixCommand(fallbackMethod = "myFallbackMethod")  5. The properties of fallback are configured in the application.properties, following the underlying syntax  hystrix.command.<groupKey>.<commandKey>.<the-property> |
| The circuit breaker gets trigged whenever the DB call duration exceeds 3 seconds. This is simulated using ThreadLocalRandom which invokes thread.sleep() with random values. Whenever the value exceeds the threshold time of 3 sec, the fallback method is called which fetches the data from a cache.  If for some reason even cache isn’t having any records, dummy record with id of 0 will be displayed (to indicate some failure) |

|  |
| --- |
| **Requirement#4**  Intercept the incoming request to validate the user id is only numeric or else return appropriate validation error. |
| Created a custom exception “IncorrectInputException” to validate if the user id is numeric.  There are also other validations for mandatory field check, min and max value checks.  Eg. We have specified that the value of id should be between 1 and 1000  There are also other exceptions like ResourceNotFoundException whenever user tries to fetch a record that doesn’t exist. |

|  |
| --- |
| **Requirement#5**  Implement the entry/exit logging for the project |
| Imported the below apis, after adding the log4j dependencies  **import** org.apache.logging.log4j.LogManager;  **import** org.apache.logging.log4j.Logger;  # Made use of log4j is configured to print logs in a file  # The log path "./logs/UserService.log" is configured inside log4j2.xml  # Used @PostConstruct and @PreDestroy to log startup and shutdown, respectively |

|  |
| --- |
| **Requirement#6**  Write the unit, Integration and pact testing(validate the contracts) for it  (Create your own pact contract json for validation, By Assuming the contract is provided by the consumer) |
| 1) Failure to Insert 5 to 10 users in h2 db during the start-up  Expected result:  a) If file not found, ERROR needs to be logged with appropriate reason  b) If there are any other reasons for failure, ERROR needs to be logged with appropriate reason  2) Prevent insertion of duplicate records in h2 db during the start-up (through users.json file)  Expected result:  If same “id” is repeated in the users.json file, the latter record should over-write the former  3) DB call response to fetch the list of users, takes more than 3 seconds  Expected result: Fallback Mechanism  a) Fetch the data from the cache, if data present in cache  b) Return empty record to represent no data could be retrieved  4) Perform field validation  Expected result: All fields are mandatory and the values need to be valid  a) HTTP response 200, if data is good  b) HTTP response 400, for invalid data  Unit testing done using restTemplate calls  End-to-end testing done through POSTMAN / Curl |

|  |
| --- |
| **Requirement#7**  Perform Basic User authentication and authorization for these end points.  Would be great if there are multiple roles(not mandatory)  Eg: Admin user should be able to update user details  Non-admin user can only access user details API (read only access). |
| # Implemented Spring Security to perform  a) DB authentication  b) Role based authorization  c) URL based authorization  # Authentication is basic DB authentication wherein, the password is encoded using bcrypt password encoder (which uses one-way hashing to save the passwords)  # Configured user named “admin” with “ADMIN” role who should be able to update user details, using HTTP PUT operation  # Configured user named “genUser” with “USER” role who should be able to update user details, using HTTP PUT operation  # Configured other end-points like the “Home Page” which can be accessed by any user without authentication |

**Key Dependencies**

1) Spring Boot Dev Tools

2) Lambok

3) Spring Data JPA

4) H2 Database

5) Spring Security

6) Spring Web

7) Hystrix [Maintenance]

8) Hystrix Dashboard [Maintenance]

**Packaging Running the application**

To build and package a Spring Boot app into a single executable Jar file with a Maven, use the below command

**maven package** (or) **mvn install**

To run your Spring Boot app from a command line in a Terminal window you can you the java -jar command.

**java -jar target/UserService 0.0.1-SNAPSHOT.jar**

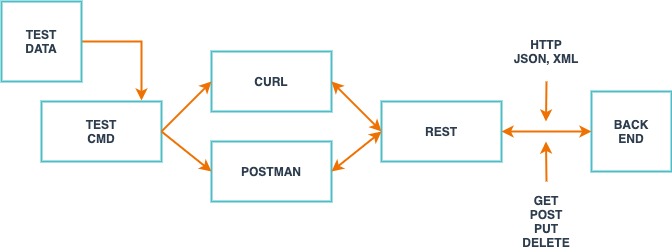
To run Spring Boot app using Maven, use the below command

**mvn spring-boot:run**

**Testing the application From Browser**

|  |
| --- |
| **Circuit Breaker Pattern:**  Normal Scenario: Fetch all data from the DB  Fallback Scenario:  a) Fetch the data from the cache, if DB isn’t accessible but data is present in cache  b) Return empty record to represent no data could be retrieved, from cache too |
| Normal Scenario: Fetch all data from the DB    Data from logs |
| Fetch the data from the cache, if DB isn’t accessible but data is present in cache    Data from logs |
| Return empty record to represent no data could be retrieved from cache too    Data from logs |

|  |
| --- |
| **Basic API Validations** |
| Get User Details : Endpoint <http://localhost:8080/api/users/4>  HTTP Operation: GET    **Data in the database** |
| Update User Details : Endpoint http://localhost:8080/api/users/  HTTP Operation: PUT    **Data in the database** |



**Testing the application using Postman UI**

Covered as part of the document 🡪 “Contract – UserService.docx”

**Testing the application using Curl**

curl --user "genUser" http://localhost:8080/api/v1/users

C:\Users\star>curl --user "admin" --request PUT --data '{\"id\":2,\"title\":\"mrs\",\"firstName\":\"UpdateTest1\",\"lastName\":\"Updatelast2\",\"gender\":\"female\",\"address\":{\"street\":\"holling\",\"city\":\"Melbourne\",\"state\":\"VIC\",\"postcode\":\"3000\"}}' <http://localhost:8080/api/v1/users>

