**WEEK – 05**

**Microservices with Spring Boot 3 and Spring Cloud**

**Superset ID: 6373188**

**Exercise 1:**

**Creating Microservices for account and loan**

**SOLUTION:**

In this hands on exercises, we will create two microservices for a bank. One microservice for handing accounts and one for handling loans.  
  
Each microservice will be a specific independent Spring RESTful Webservice maven project having it's own pom.xml. The only difference is that, instead of having both account and loan as a single application, it is split into two different applications. These webservices will be a simple service without any backend connectivity.

**AccountApplication.java :**

*package com.example.account;*

*import org.springframework.boot.SpringApplication;*

*import org.springframework.boot.autoconfigure.SpringBootApplication;*

*@SpringBootApplication*

*public class AccountApplication {*

*public static void main(String[] args) {*

*SpringApplication.run(AccountApplication.class, args);*

*}*

*}*

**Explanation:**

• This is the **main class** of the Spring Boot application.

• @SpringBootApplication is a convenience annotation that includes:

• @Configuration – for Java-based configuration

• @EnableAutoConfiguration – to auto-configure Spring settings

• @ComponentScan – to scan components in the current package

• SpringApplication.run(...) bootstraps the application.

**AccountController.java:**

*package com.example.account;*

*import org.springframework.web.bind.annotation.\*;*

*@RestController*

*@RequestMapping("/accounts")*

*public class AccountController {*

*@GetMapping("/{number}")*

*public Account getAccount(@PathVariable String number) {*

*return new Account(number, "savings", 234343);*

*}*

*// Dummy response class*

*static class Account {*

*public String number;*

*public String type;*

*public double balance;*

*public Account(String number, String type, double balance) {*

*this.number = number;*

*this.type = type;*

*this.balance = balance;*

*}*

*}*

*}*

**EXPLANATION:**

• @RestController: Defines this class as a RESTful controller that returns data directly in JSON format.

• @RequestMapping("/accounts"): Prefixes all APIs with /accounts.

• @GetMapping("/{number}"): Binds GET requests like /accounts/00987987973432 to the getAccount() method.

• The method returns an Account object with dummy data (no DB connection).

**pom.xml:**

<project ...>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

</parent>

<groupId>com.example</groupId>

<artifactId>account</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>account</name>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Web dependency to create REST APIs -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- DevTools for auto-reload during development -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- Testing support -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**EXPLANATION:**

• Defines a Maven project configuration.

• Uses Spring Boot Starter Parent to simplify setup.

• Adds key dependencies:

• spring-boot-starter-web: for building RESTful APIs.

• spring-boot-devtools: for auto-restart during development.

• spring-boot-starter-test: for unit and integration testing.

• Includes Maven plugin to package and run Spring Boot app.

**OUTPUT:**

**What this Account Microservice does??**

• I created a RESTful web service using Spring Boot as part of my microservices learning project.

• I set up the complete application structure using Spring Initializr and managed the build with Maven.

• I developed and exposed an API endpoint:

• GET /accounts/{number}

• This endpoint returns dummy account details in JSON format based on the input account number.

• There is no backend database or business logic involved this was intentionally kept simple as a learning demo.

• I successfully launched the service independently and tested it in the browser at:  
http://localhost:8080/accounts/12345

• This microservice is fully standalone and ready to be integrated with other services such as a Loan Service running on a different port.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**