**Student Name:** KISHORE S

**Registration No:** 22CSR100

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**CREATING MICROSERVICES FOR ACCOUNT AND LOAN**

**Introduction:**

In today’s fast-paced software industry, building modular and scalable systems is essential. Microservices architecture helps achieve this by dividing a large application into smaller, independent services. In this hands-on project, we build two separate Spring Boot microservices — one to manage account details and the other for loan details. Both services run independently and expose RESTful endpoints with dummy data, illustrating the core concept of service independence and communication readiness.

**Objective:**

* To understand the fundamentals of microservices using Spring Boot.
* To implement two independent REST APIs — one for account service and another for loan service.
* To manage port conflicts and run multiple services simultaneously without backend/database integration.

**Implementation Breakdown:**

**STEP 1: Setup Project Folder**

1. Open **D: drive** and create a folder with your **employee ID** (e.g., 22CSR100).
2. Inside that folder, create a subfolder named **microservices**.
3. This folder will hold both microservice projects.

**STEP 2: Create Account Microservice**

1. Open <https://start.spring.io/>.
2. Fill in the following:
   * **Group**: com.cognizant
   * **Artifact**: account
3. Select dependencies:
   * **Spring Boot DevTools**
   * **Spring Web**
4. Click **Generate**, download the ZIP, and extract it.
5. Move the account folder to your microservices directory.
6. Open a terminal inside the account folder and run:

mvn clean package

1. Import the project into **Intellij IDE**.
2. Create the controller class:  
   src/main/java/com/cognizant/account/AccountController.java

**AccountController.java:**

package com.cognizant.account.controller;

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

import java.util.Map;

@RestController

@RequestMapping("/accounts")

public class AccountController {

@GetMapping("/{accountNumber}")

public Map<String, Object> getAccountDetails(@PathVariable String accountNumber) {

return Map.of(

"accountNumber", accountNumber,

"accountType", "Savings",

"currentBalance", 234343

);

}

}

1. Run AccountApplication.java and test in browser:

<http://localhost:8080/accounts/00987987973432>

**STEP 3: Create Loan Microservice**

1. Repeat the same process from Spring Initializr:
   * **Group**: com.cognizant
   * **Artifact**: loan
2. Select dependencies:
   * **Spring Boot DevTools**
   * **Spring Web**
3. Extract and move the loan folder to microservices.
4. Open terminal in loan and build:

mvn clean package

1. Import into **Intellij IDE**.
2. Add a different port to avoid conflict with Account service:  
   **src/main/resources/application.properties**

server.port=8081

1. Create controller:  
   src/main/java/com/cognizant/loan/LoanController.java

**LoanController.java:**

package com.cognizant.loan.controller;

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

import java.util.Map;

@RestController

@RequestMapping("/loans")

public class LoanController {

@GetMapping("/{loanNumber}")

public Map<String, Object> getLoanDetails(@PathVariable String loanNumber) {

return Map.of(

"loanNumber", loanNumber,

"loanType", "Car Loan",

"loanAmount", 400000,

"monthlyEMI", 3258,

"tenureInMonths", 18

);

}

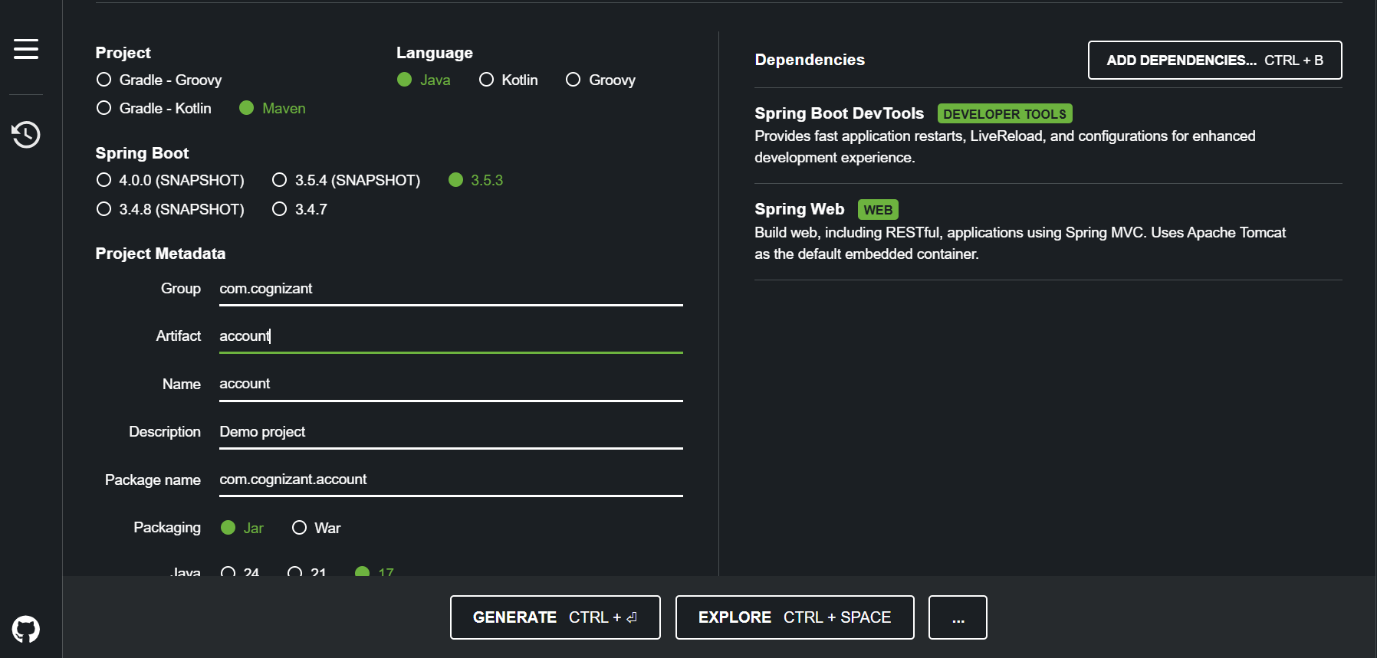
}

1. Run LoanApplication.java and test:

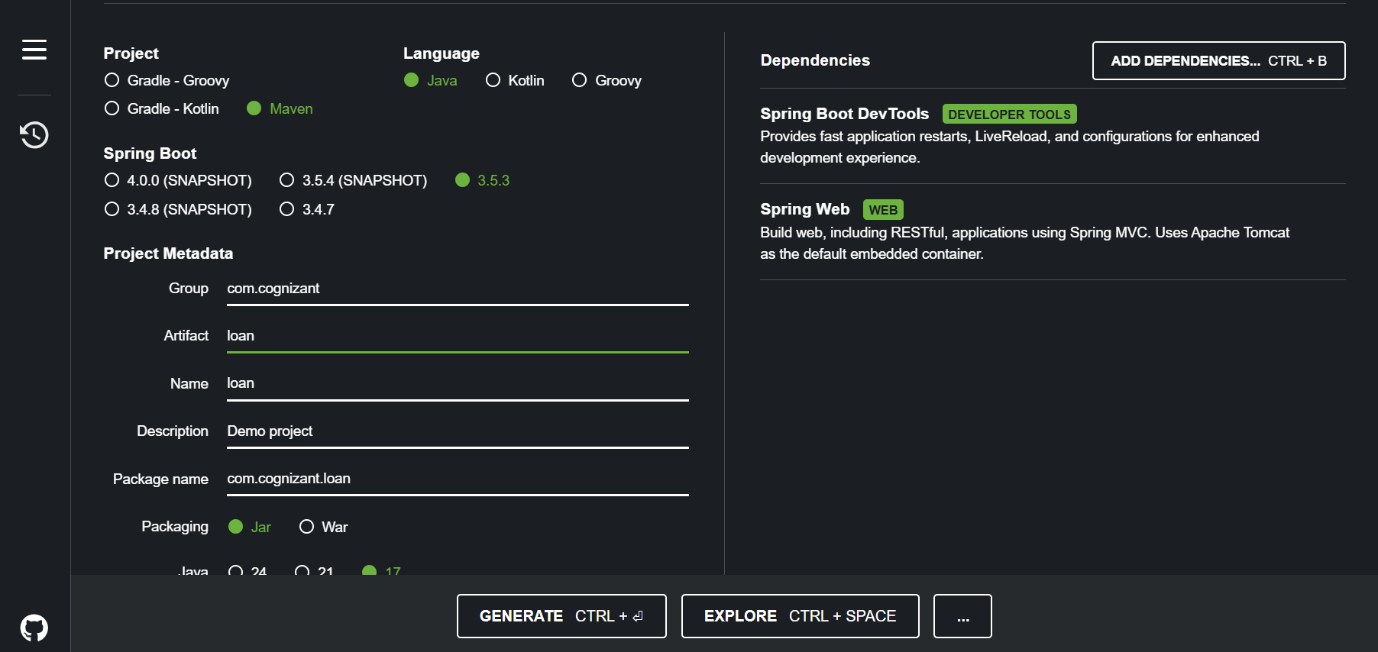
http://localhost:8081/loans/H00987987972342

**Output:**

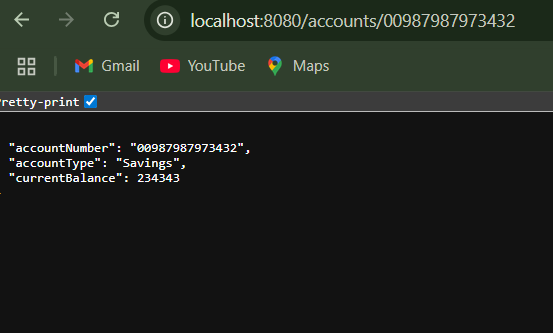
**Spring Initializer(Account):**

****

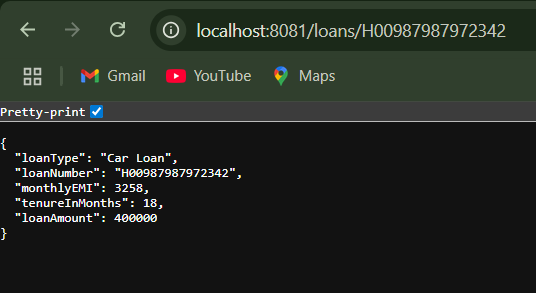
**Spring Intializer(Loan):**

****

**Account:**

****

**Loan:**

****

**Conclusion:**

By completing this mini-project, I’ve taken my first step into the world of microservices. I can now understand how to break a monolithic application into independent services, each with its own responsibility, configuration, and port. Although the services here return static responses, they form the foundation for scalable, database-connected, and cloud-ready microservices in the future.