# **WEEK 5 microservices with spring boot 3 and spring cloud HANDS ON**

**SUBMITTED BY :-**

NAME – AYUSH SAHOO

SUPERSET ID – 6372243

COURSE – DIGITAL NURTURE 4.0 JAVA FSE

EMAIL – [ayush.sahoo003@gmail.com](mailto:ayush.sahoo003@gmail.com)

COLLEGE – C.V. RAMAN GLOBAL UNIVERSITY

**-----------------NOTE: I HAVE ALSO DONE ADDITIONAL IMPORTANT HANDS ON -------------------**

**MANDATORY HANDS ON**

**EXERCISE 1:**

**Creating Microservices for account and loan**

**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.**

**Follow steps below to implement the two microservices:**

**Account Microservice**

**• Create folder with employee id in D: drive**

**• Create folder named 'microservices' in the new folder created in previous step. This folder will contain all the sample projects that we will create for learning microservices.**

**• Open https://start.spring.io/ in browser**

**• Enter form field values as specified below:**

**o Group: com.cognizant**

**o Artifact: account**

**• Select the following modules**

**o Developer Tools > Spring Boot DevTools**

**o Web > Spring Web**

**• Click generate and download the zip file**

**• Extract 'account' folder from the zip and place this folder in the 'microservices' folder created earlier**

**• Open command prompt in account folder and build using mvn clean package command**

**• Import this project in Eclipse and implement a controller method for getting account details based on account number. Refer specification below:**

**o Method: GET**

**o Endpoint: /accounts/{number}**

**o Sample Response. Just a dummy response without any backend connectivity.**

**{ number: "00987987973432", type: "savings", balance: 234343 }**

**• Launch by running the application class and test the service in browser**

**Loan Microservice**

**• Follow similar steps specified for Account Microservice and implement a service API to get loan account details**

**o Method: GET**

**o Endpoint: /loans/{number}**

**o Sample Response. Just a dummy response without any backend connectivity.**

**{ number: "H00987987972342", type: "car", loan: 400000, emi: 3258, tenure: 18 }**

**• Launching this application by having account service already running**

**• This launch will fail with error that the bind address is already in use**

**• The reason is that each one of the service is launched with default port number as 8080. Account service is already using this port and it is not available for loan service.**

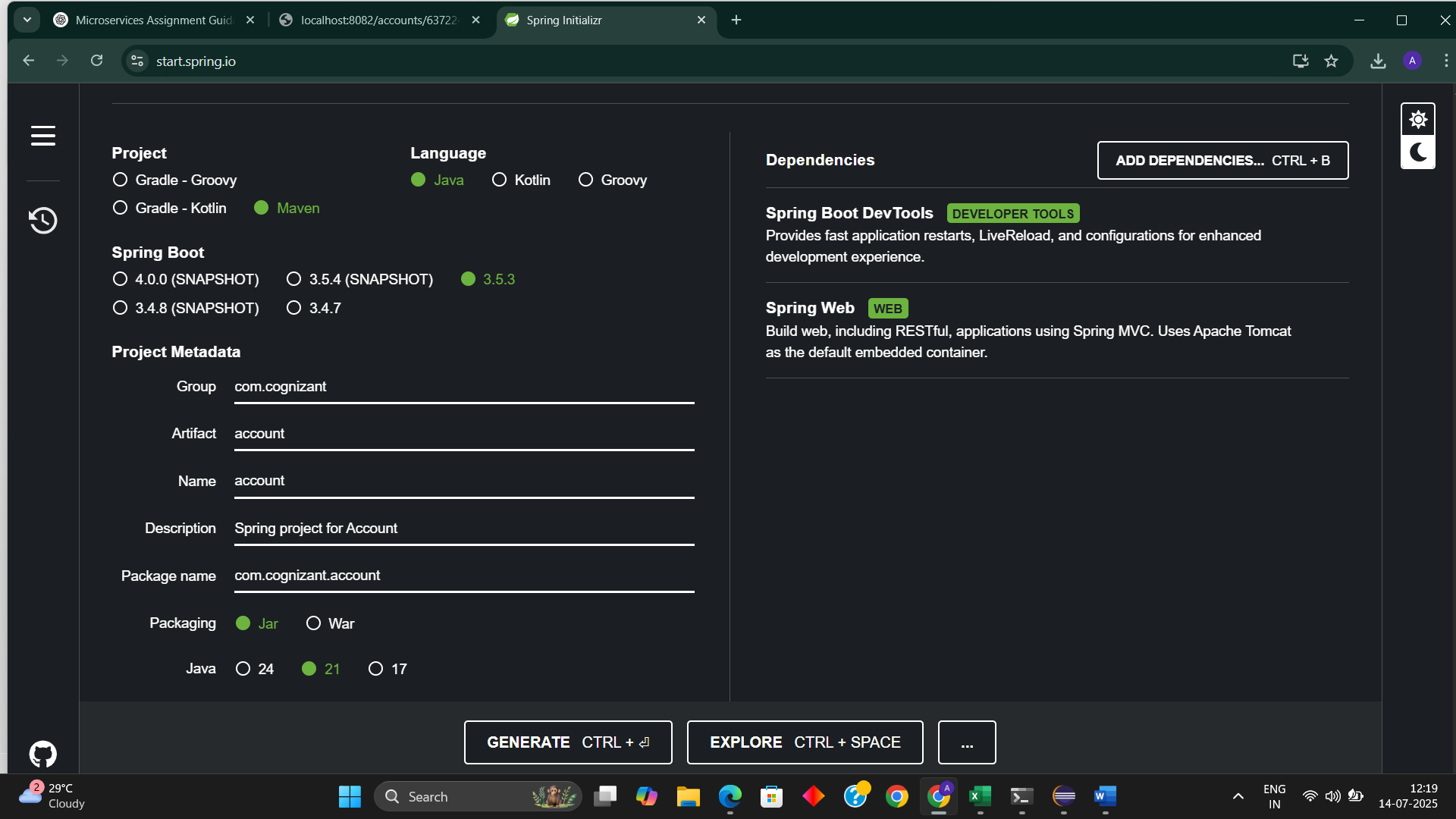
**• Include "server.port" property with value 8081 and try launching the application**

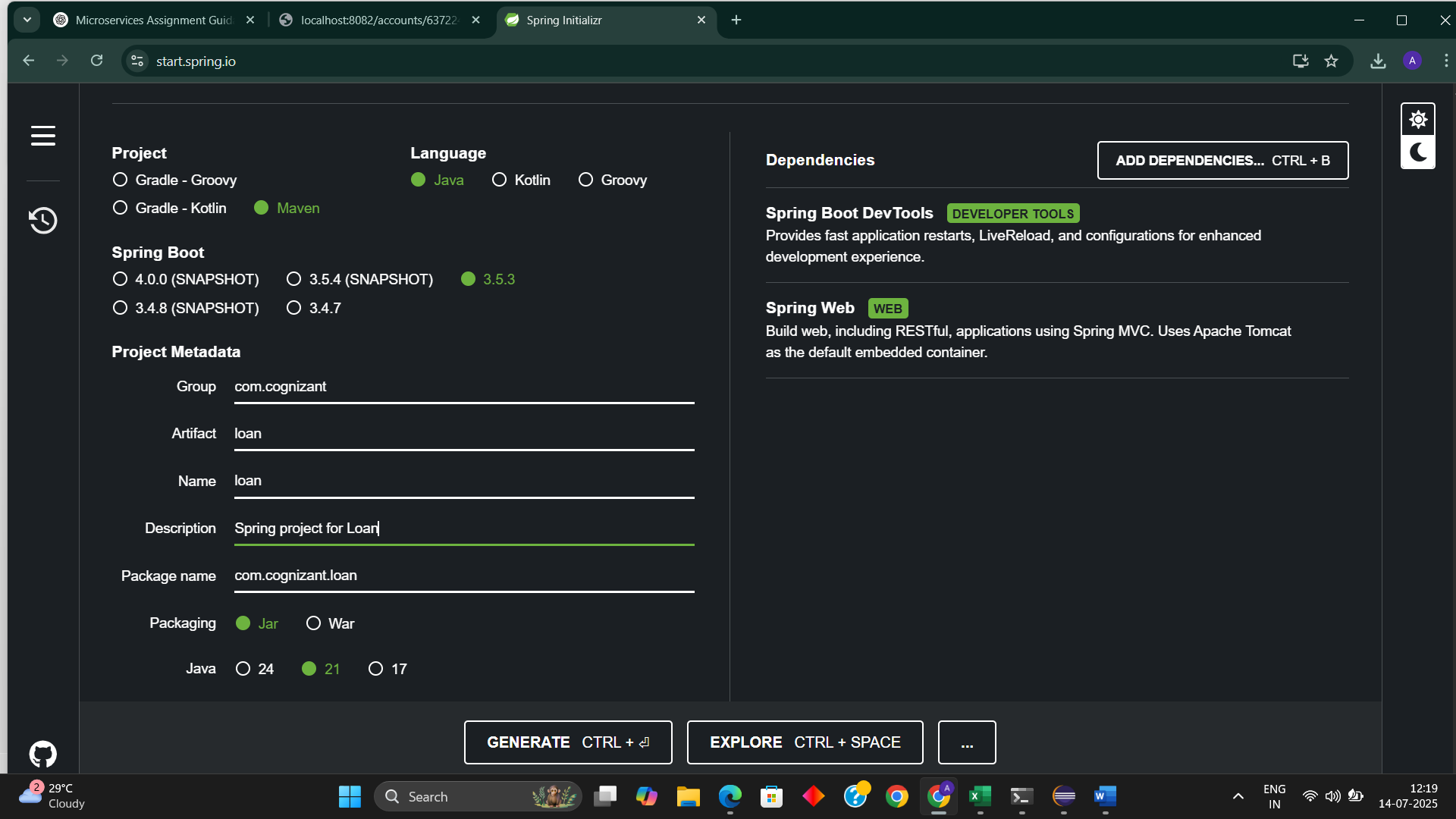
**• Test the service with 8081 port Now we have two microservices running on different ports.**

**NOTE: The console window of Eclipse will have both the service console running. To switch between different consoles use the monitor icon within the console view.**

**SOLUTION:-**

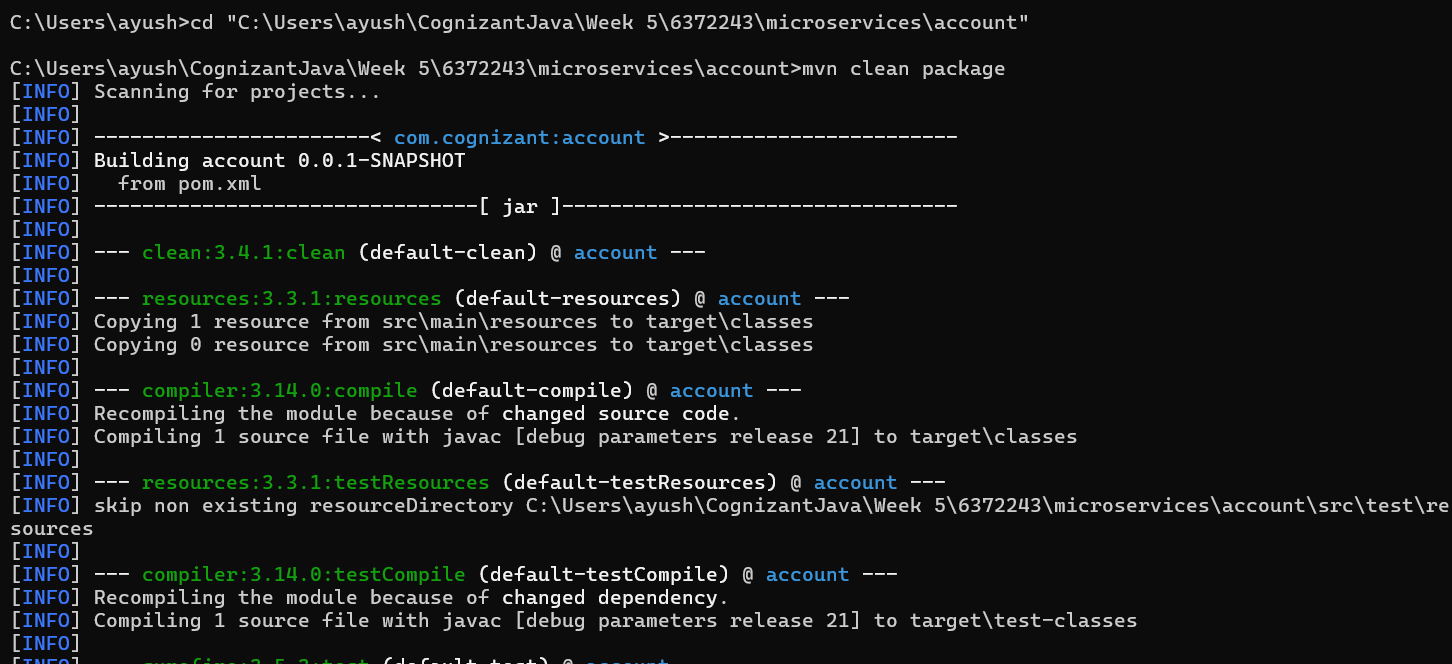
Step 1: Create a zip file for account and loan (via spring initializr) (account and loan respectively)



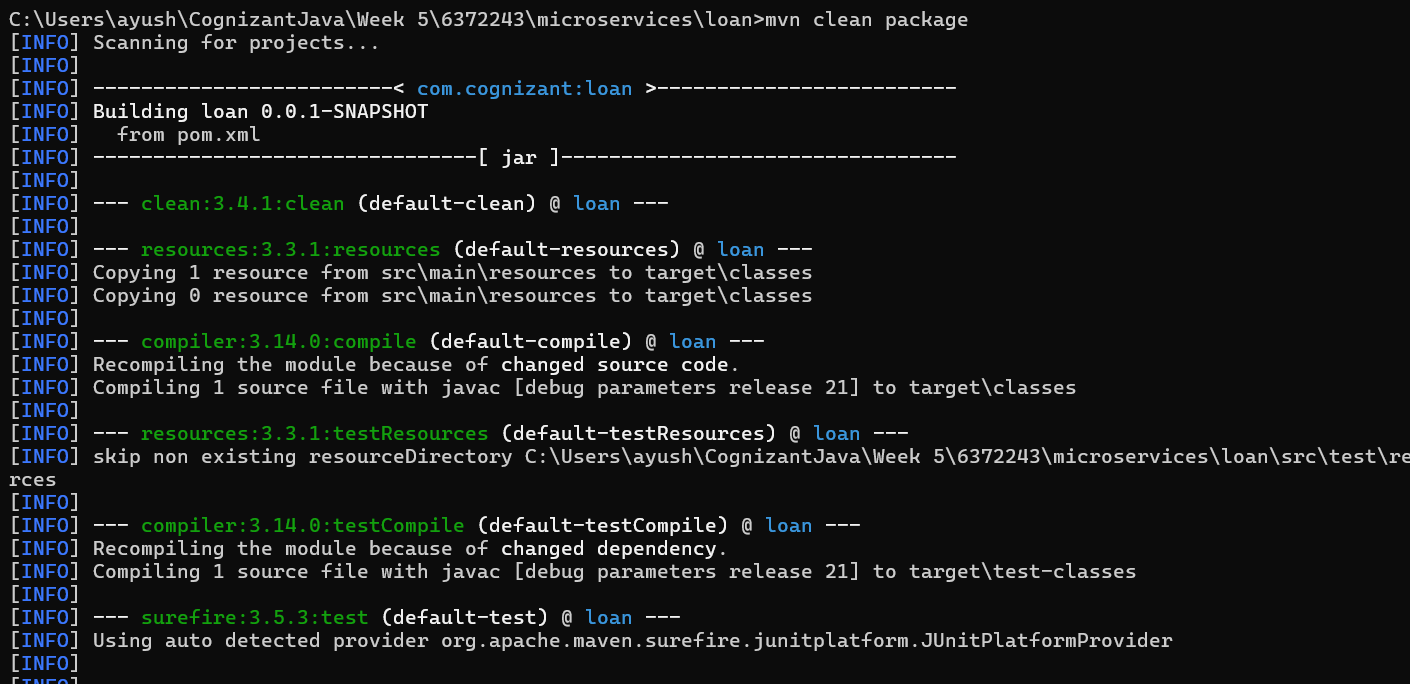


Step 2: To navigate each project and build **(Here instead of random employee number I have taken my superset Id as employee number)**

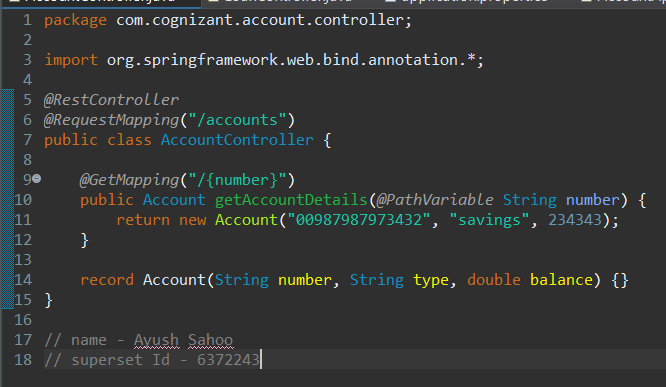
**For Account**



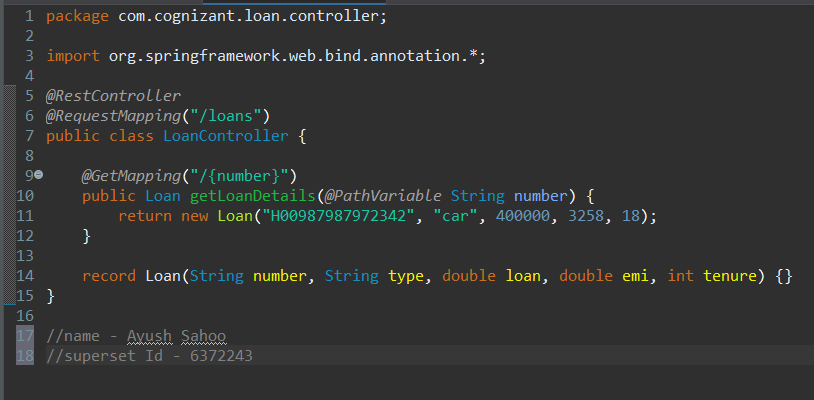
**For Loan**

****

Step 3: Create a new class as “AccountController.java” inside package com.cognizant.account.controller.

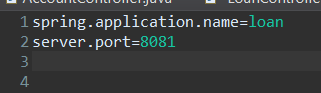


Step 4: Create a new class “LoanController.java” inside package com.cognizant.loan.controller

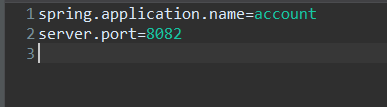


Step 6: Change port for both loan and account in microservices (as port 8080 is already in use for different project)

* For loan

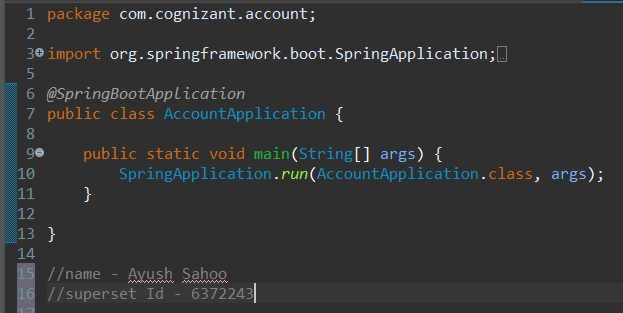


* For Account

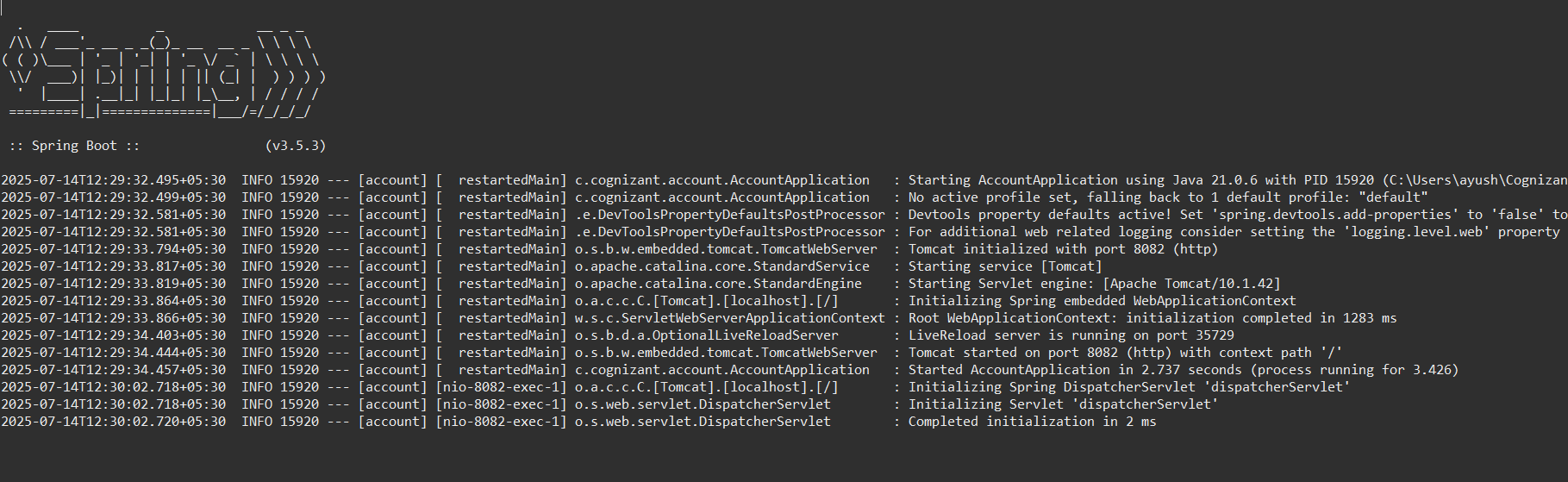


**OUTPUT:**

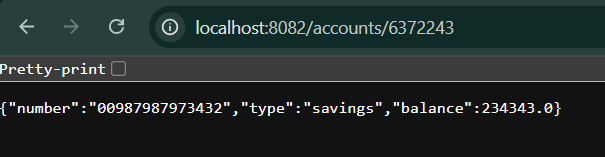
**For Account:** Run “AccountApplication.java”.

****

Eclipse Console:

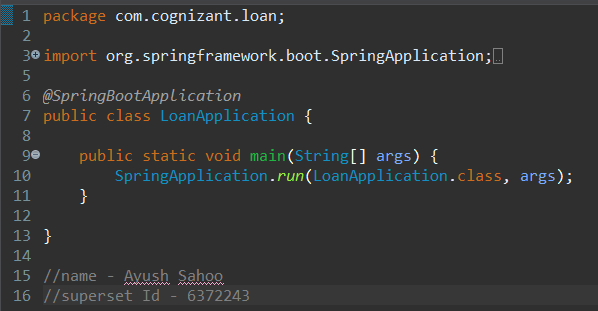


Browser Result:

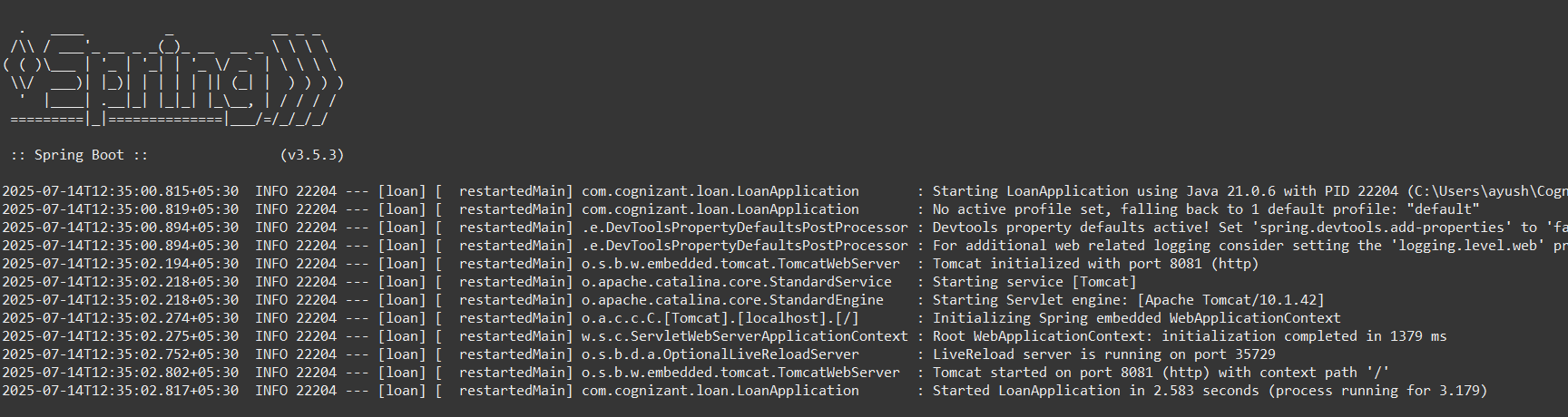


**For Loan:**

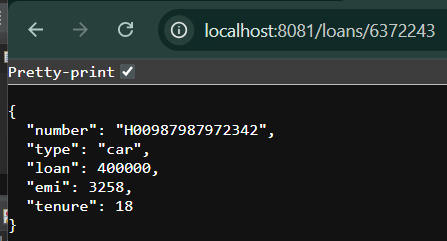
Run “LoanApplication.java”.

****

Eclipse console:



Browser Result:



ADDITIONAL HANDS ON

**EXERCISE 1:**

**Create Eureka Discovery Server and register microservices**

**Eureka Discovery Server holds a registry of all the services that are available for immediate consumption. Anybody whom wants to consume a RESTful Web Service can come to the discovery server and find out what is available and ready for consumption. Eureka Discovery Server is part of spring cloud module.**

**Follow steps below to implement:**

**Create and Launch Eureka Discovery Server**

**Using https://start.spring.io generate a project with following configuration:**

**o Group: com.cognizant**

**o Artifact: eureka-discovery-server**

**o Module: Spring Cloud Discovery > Eureka Server**

**• Download the project, build it using maven in command line**

**• Import the project in Eclipse**

**• Include @EnableEurekaServer in class EurekaDiscoveryServerApplication**

**• Include the following configurations in application.properties:**

**server.port=8761**

**eureka.client.register-with-eureka=false**

**eureka.client.fetch-registry=false**

**logging.level.com.netflix.eureka=OFF**

**logging.level.com.netflix.discovery=OFF**

**• The above configuration runs the discovery service in port 8761**

**• The eureka properties prohibits direct registration of services, instead discovery server will find available services and register them.**

**• Launch the service by running the application class**

**• The discovery service can be view by launching http://locahost:8761 in the browser.**

**• This will display the discover server details**

**• Look into the section "Instances currently registered with Eureka", which will have an empty list • Follow steps below to add account and loan service to this discovery server.**

**Register Account REST API to eureka discovery**

**• Go to https://start.spring.io and provide the following configuration:**

**o Group: com.cognizant**

**o Artifact: account**

**o Modules:**

**♣ Spring Boot DevTools**

**♣ Eureka Discovery Client**

**♣ Spring Web**

**• Click "Explore", which will open pom.xml**

**• Use copy option in the opened window to copy the pom.xml and overwrite the pom.xml in account project**

**• Build the project using maven in console**

**• Include @EnableDiscoveryClient annotation to application class of account project**

**• Include application name for account application as specified below in application.properties. This is the name that will be displayed in the eureka discovery registry.**

**spring.application.name=account-service**

**• Stop all services (account, loan, eureka-discovery-server) using the console window of Eclipse. Use the monitor icon in console view to switch between applications and use the Terminate button to stop the server.**

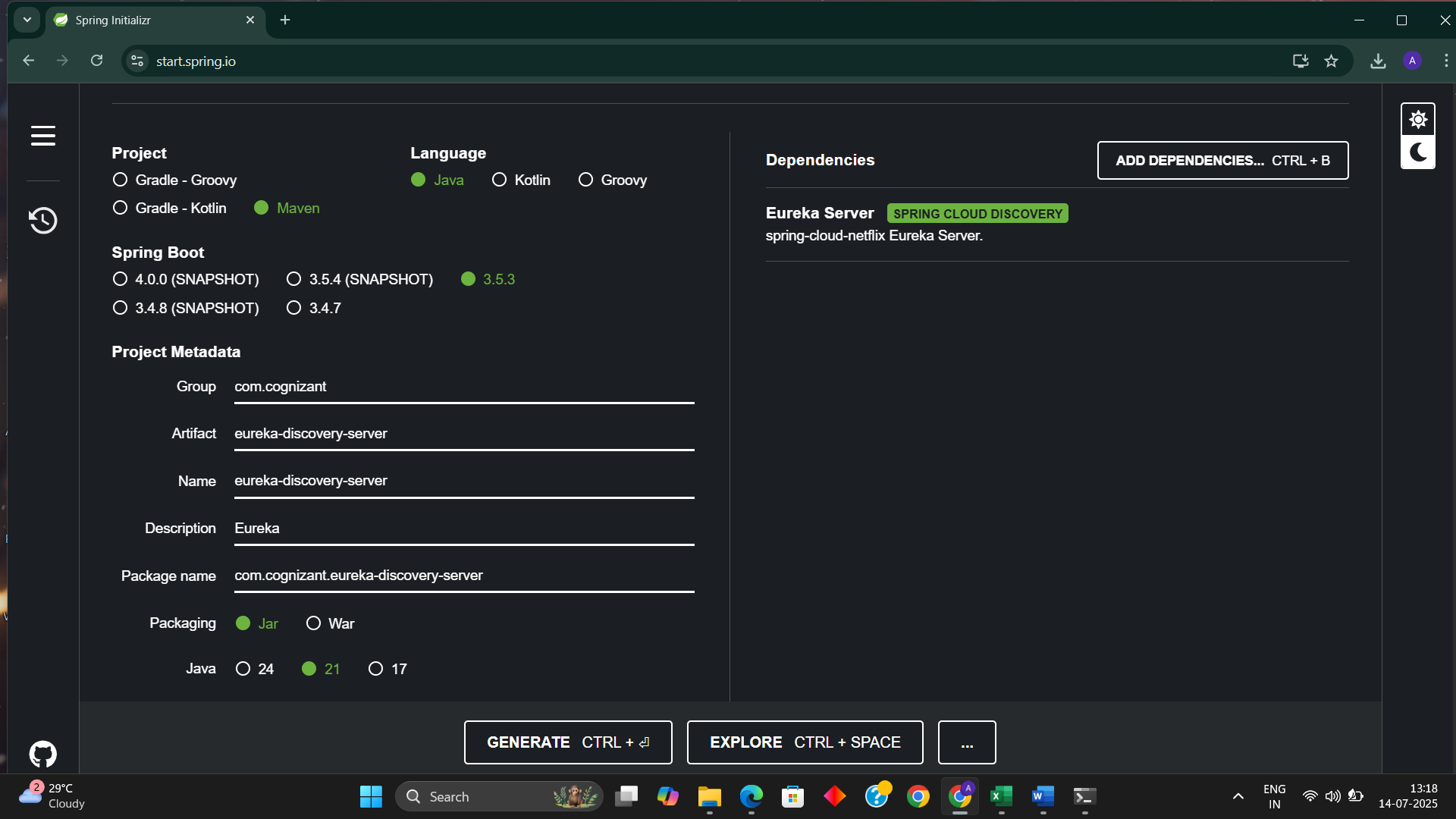
**• First start eureka-discovery-server and wait till the application starts completely. Then open http://locahost:8761 in browser. The service list should be empty.**

**• Then start account application and wait till the application starts.**

**• Refresh the eureka-discovery-server web page in browser, the accountservice will be listed in the registry**

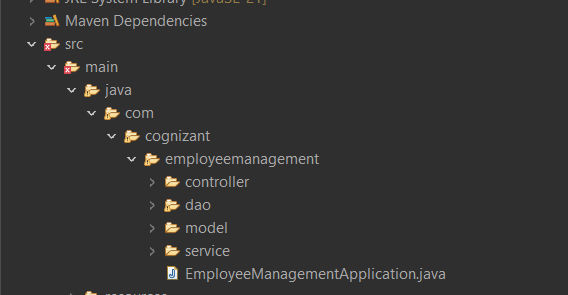
**• Perform similar steps for loan application and have it registered with eureka-discovery-server.**

**SOLUTION:**

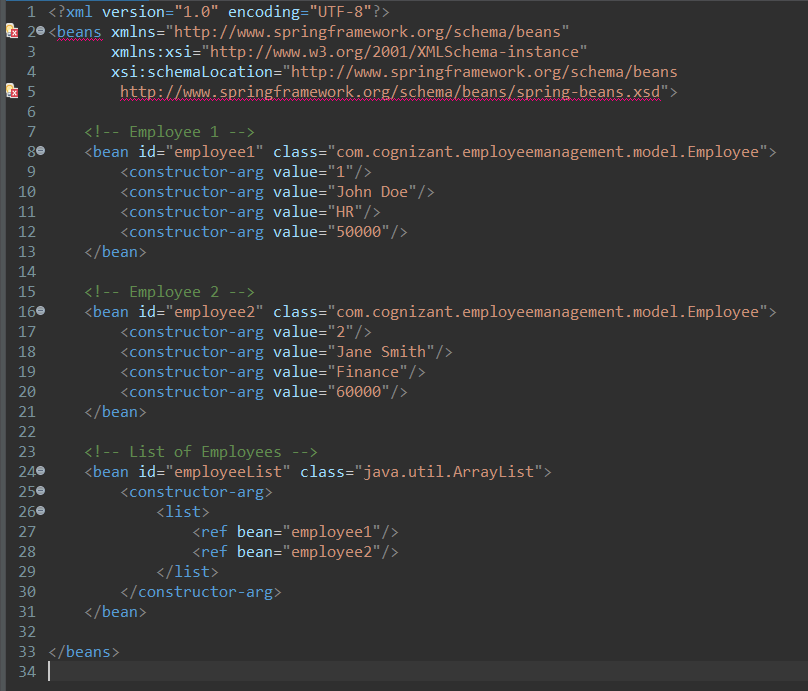
****

****

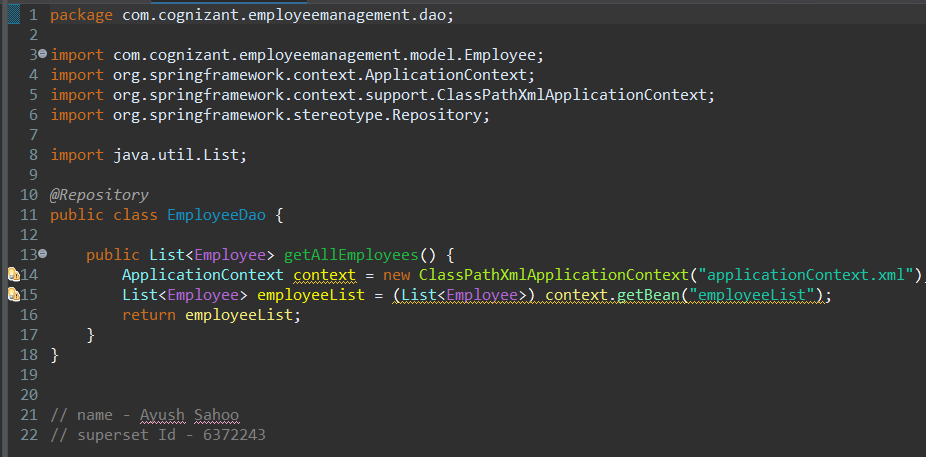
Step 1: Folder Structure



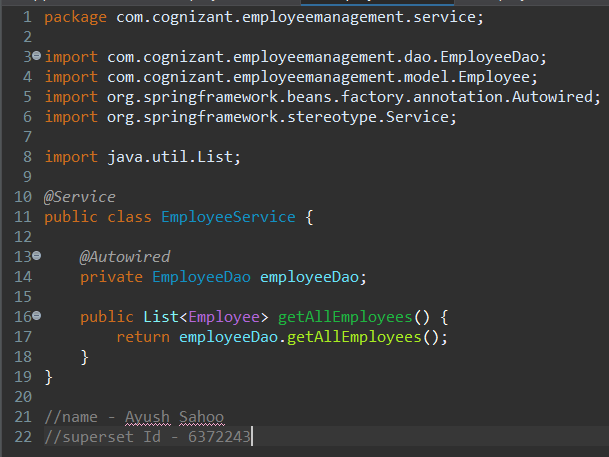
Step 2: Create a “applicationContext.xml” under src/main/resources.



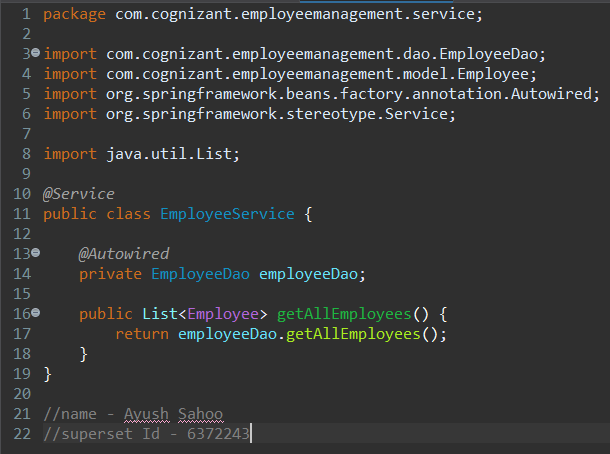
Step 3: Create a “EmployeeDao.java” under src/main/java/com/cognizant/employeemanagement/dao.



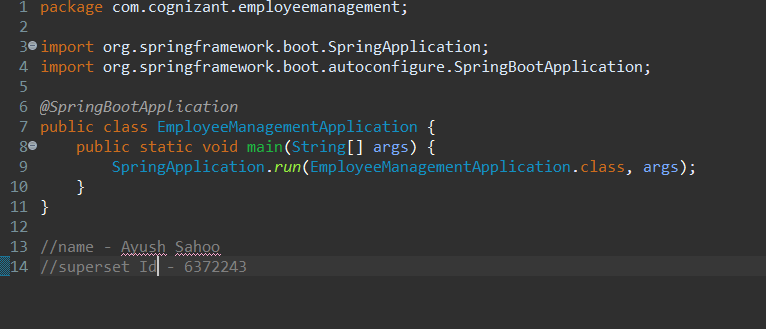
Step 4: Create a “EmployeeService.java” under src/main/java/com/cognizant/employeemanagement/service.



Step 5: Create a “EmployeeController.java” under src/main/java/com/cognizant/employeemanagement/controller

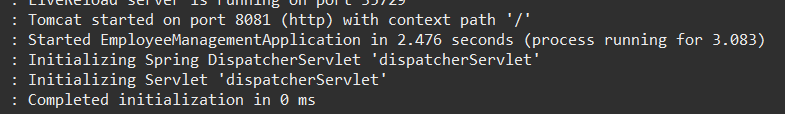


Step 6: Run “Employeemanagement.java”

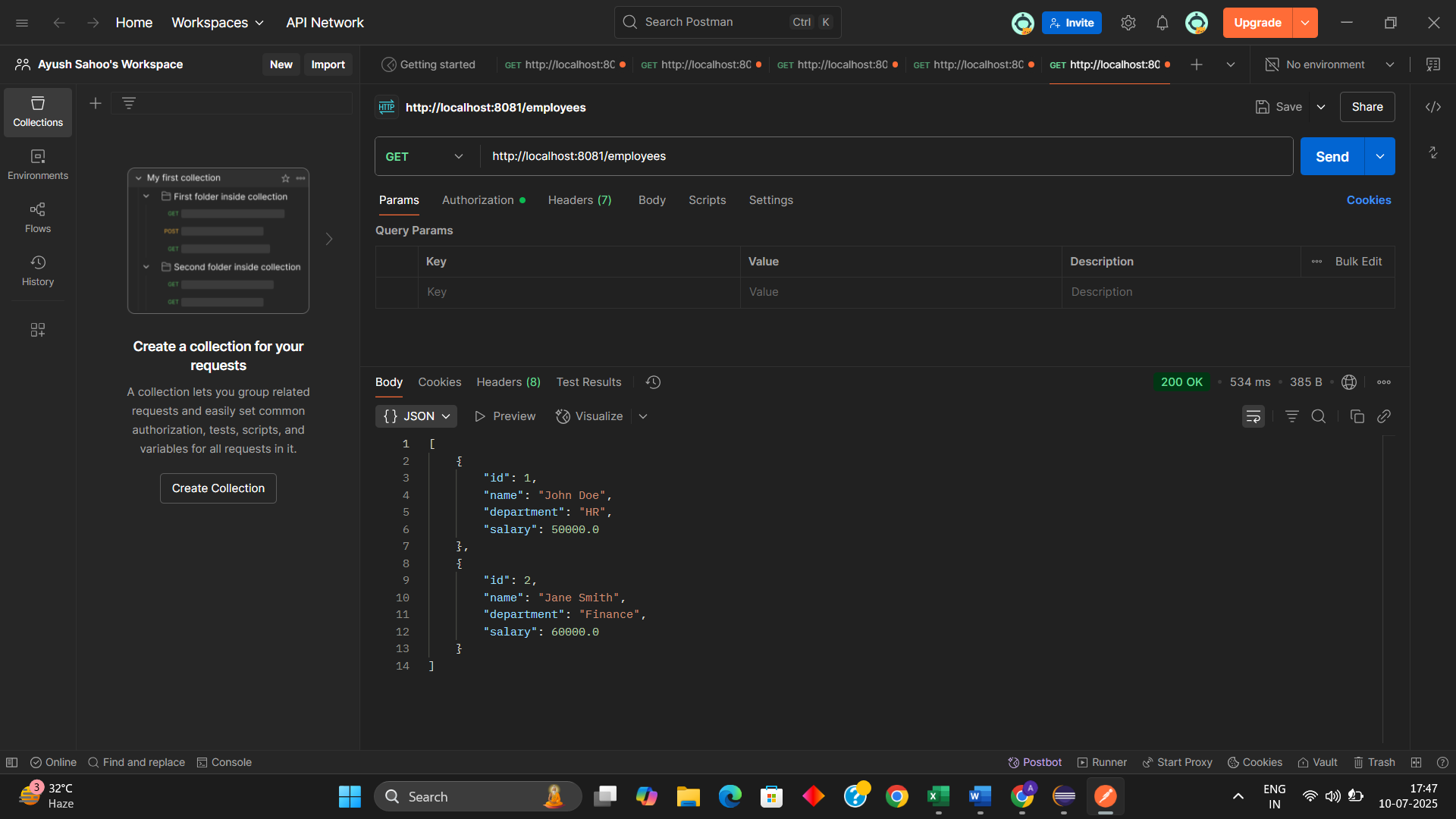


**OUTPUT:**

**Console (Of Eclipse)**

****

**(POSTMAN OUTPUT)**

****

**x----x----x**