WEEK 5

Taniya G Remula

St. Joseph's Institute of Technology

Superset ID: 6376013

Microservices with API gate

Exercise 1: Creating Microservices for account and loan

AccountServiceApplication.java:

```
package com.example.accountservice;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class AccountServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(AccountServiceApplication.class, args);
    }
}
```

AccountController.java

```
package com.example.accountservice.controller;
import com.example.accountservice.model.Account;
import com.example.accountservice.service.AccountService;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/accounts")
public class AccountController {
    private final AccountService accountService;
    public AccountController(AccountService accountService) {
        this.accountService = accountService;
    }
```

```
    @PostMapping
    public Account createAccount(@RequestBody Account account) {
        return accountService.saveAccount(account);
    }
    @GetMapping
    public List<Account> getAccounts() {
        return accountService.getAllAccounts();
    }
}
```

Account.java

```
package com.example.accountservice.model;
import jakarta.persistence.*;
@Entity
public class Account {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String accountNumber;
    private String name;
    private double balance;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getAccountNumber() {
        return accountNumber;
    }
    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }
```

```
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getBalance() {
        return balance;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }
}
```

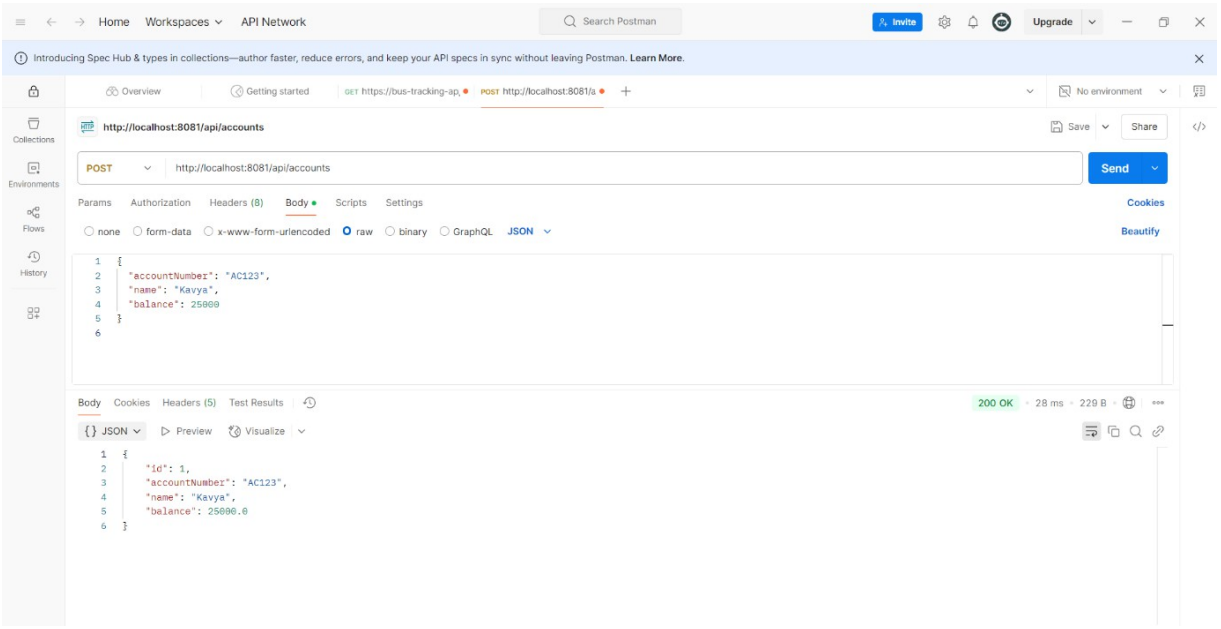
AccountRepository.java

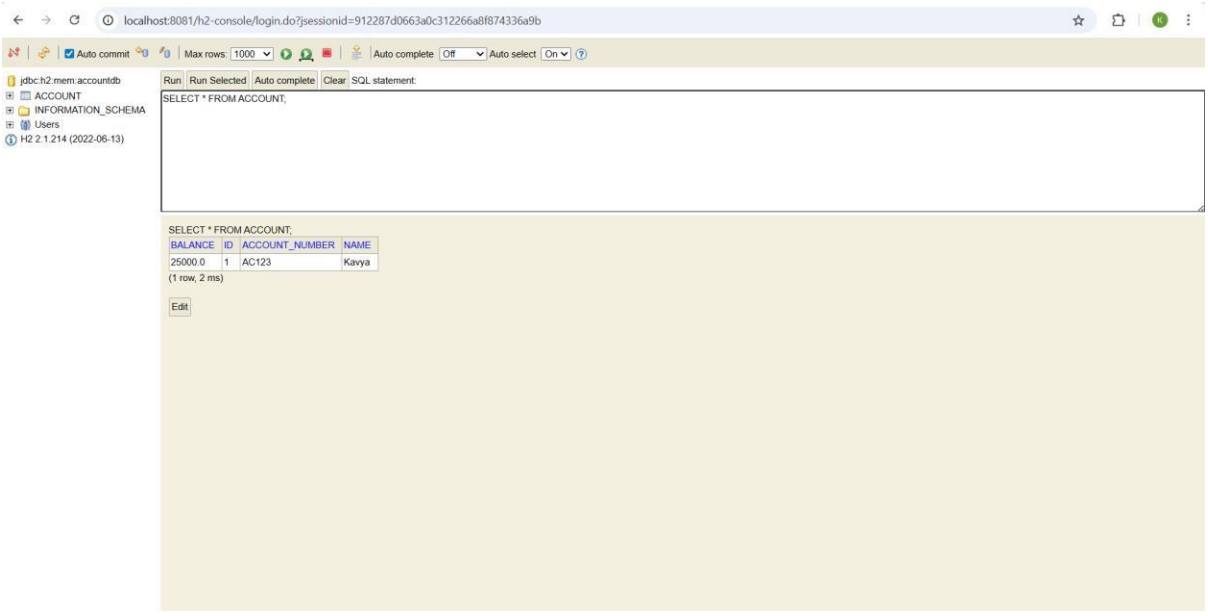
```
package com.example.accountservice.repository;
import com.example.accountservice.model.Account;
import org.springframework.data.jpa.repository.JpaRepository;
public interface AccountRepository extends JpaRepository<Account, Long> {
}
```

Application.properties

```
server.port=8081
spring.datasource.url=jdbc:h2:mem:accountdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

Output:


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Introducing Spec Hub & types in collections—author faster, reduce errors, and keep your API specs in sync without leaving Postman. Learn More.
Overview
Getting started
No environment
http://localhost:8081/api/accounts
Save
Share
Collections
Environments
Flows
History
POST
http://localhost:8081/api/accounts
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"accountNumber": "AC123",
"name": "Kavya",
"balance": 25000
}
Body
Cookies
Headers (5)
Test Results
200 OK
28 ms
229 B
JSON
Preview
Visualize
{
"id": 1,
"accountNumber": "AC123",
"name": "Kavya",
"balance": 25000.0
}


localhost:8081/h2-console/login.do?jsessionid=912287d0663a0c312266a8f874336a9b
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:accountdb
ACCOUNT
INFORMATION_SCHEMA
Users
H2 2.1.214 (2022-06-13)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM ACCOUNT;
SELECT * FROM ACCOUNT;
BALANCE
ID
ACCOUNT_NUMBER
NAME
25000.0
1
AC123
Kavya
(1 row, 2 ms)
Edit

Exercise 2: Create Eureka Discovery Server and register

EurekaDiscoveryServerApplication.java

```
package com.example.eurekadiscoveryserver; import
org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
@SpringBootApplication
@EnableEurekaServer
public class EurekaDiscoveryServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaDiscoveryServerApplication.class, args);
    }
}
```

Application.properties

```
server.port=8761
spring.application.name=eureka-discovery-server
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Output: