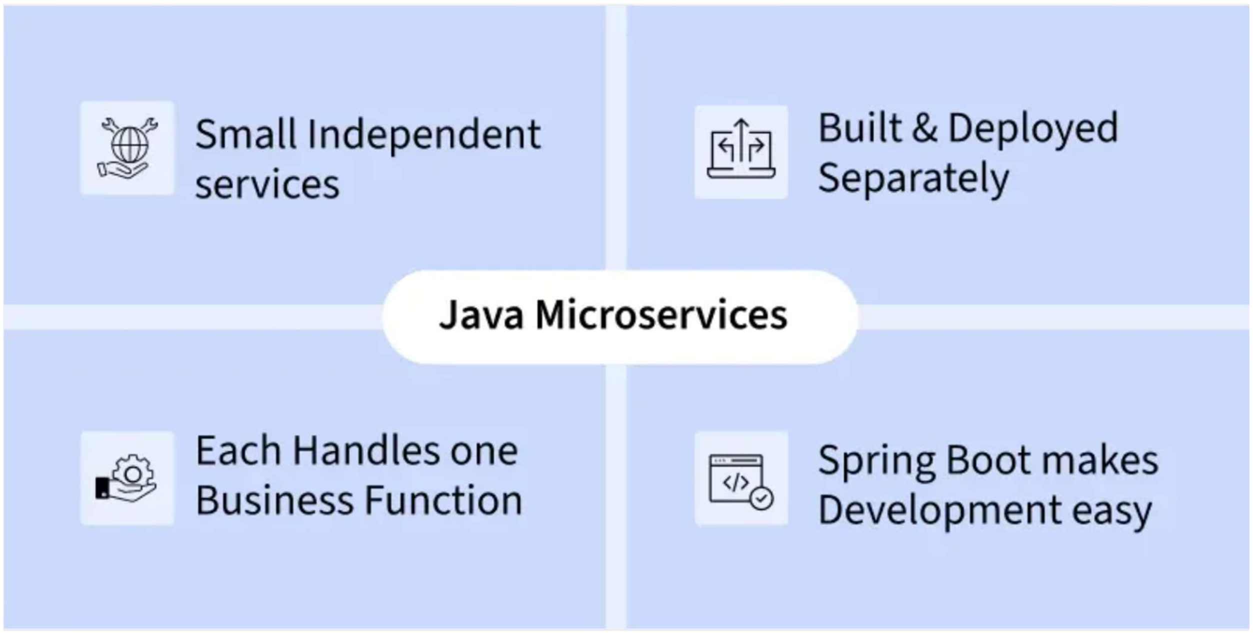
Microservices

What is Microservices?

- Microservice are **small business services** that can work together and can be **deployed autonomously / independently**
- These services **communicate with each other** by talking over the network and bring **many advantages** with them.
- One of the biggest advantages is that they can be **deployed independently.**
- However, **it offers the opportunity** to work with **many different technology.**


Small Independent services
Built & Deployed Separately
Java Microservices
Each Handles one Business Function
Spring Boot makes Development easy

How to Perform it :-

User Service (DB) + Order Service(DB) → fetched by → Report Service

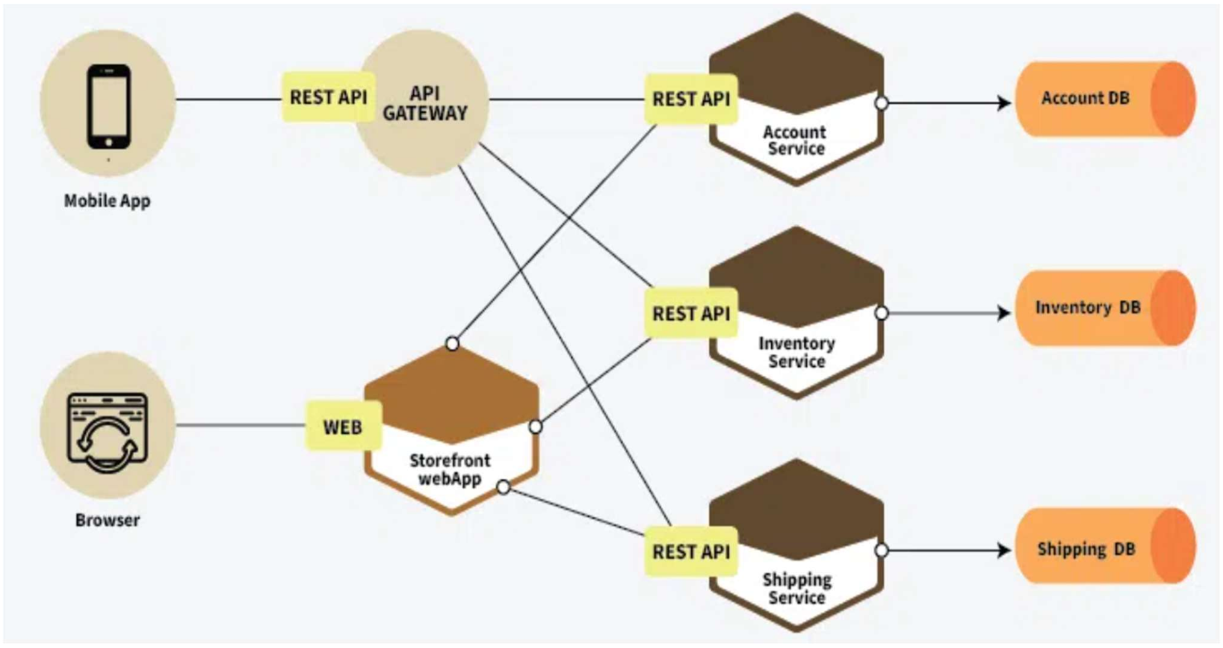
How do Microservices work?

- **Feature-based services:**
  Each microservice focuses on one specific business function (e.g., user, product).

- **API-based communication:**
  Services talk to each other using APIs, making integration easy and standardized.

- **Technology flexibility:**
  Each service can use different programming languages or tools based on what suits it best.

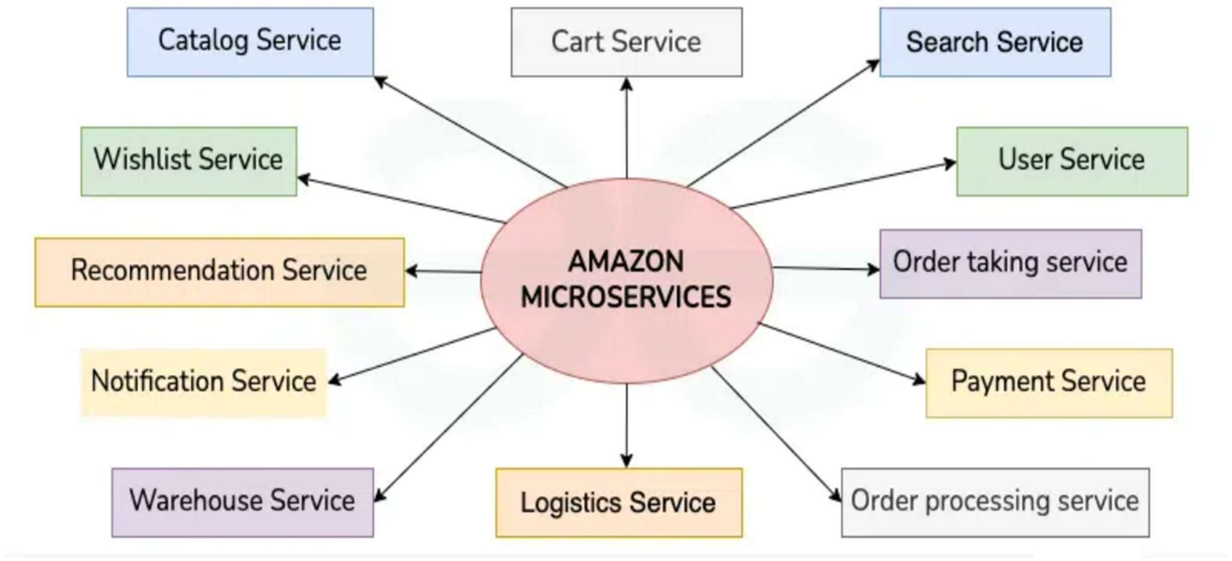
- **Independent updates:**
  Services can be changed or deployed separately, which lowers risk and improves system stability.



Architecture of Microservices

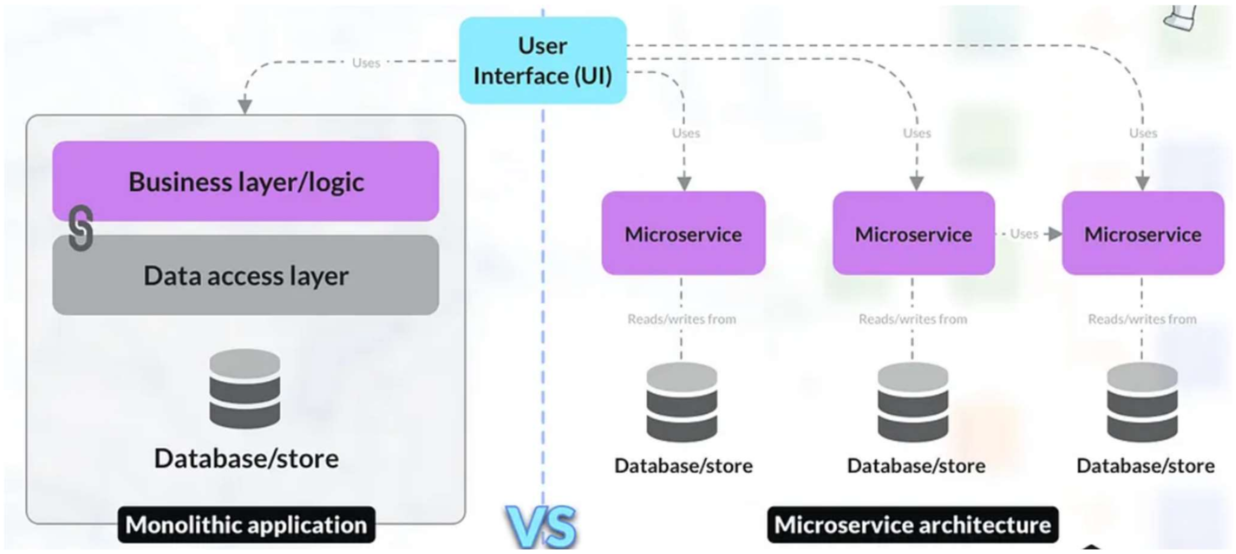
Real-World Example of Microservices :-

- Let's understand the Miscroservices using the real-world example of Amazon E-Commerce Application:



- **User Service:** Manages user accounts, profiles, and preferences for personalization.
- **Search Service:** Enables fast product search using indexed product data.
- **Catalog Service:** Maintains accurate and accessible product listings.
- **Cart Service:** Handles adding, removing, and updating items before checkout.
- **Wishlist Service:** Saves products for future purchase.
- **Order Taking Service:** Validates and places customer orders.
- **Order Processing Service:** Manages order fulfilment with inventory and shipping.
- **Payment Service:** Handles secure payment transactions.
- **Logistics Service:** Manages delivery, shipping charges, and tracking.
- **Warehouse Service:** Monitors stock levels and restocking.
- **Notification Service:** Sends order updates and promotional messages.
- **Recommendation Service:** Suggests products based on user behaviour.

What is Microservices and Monolithic architecture :-



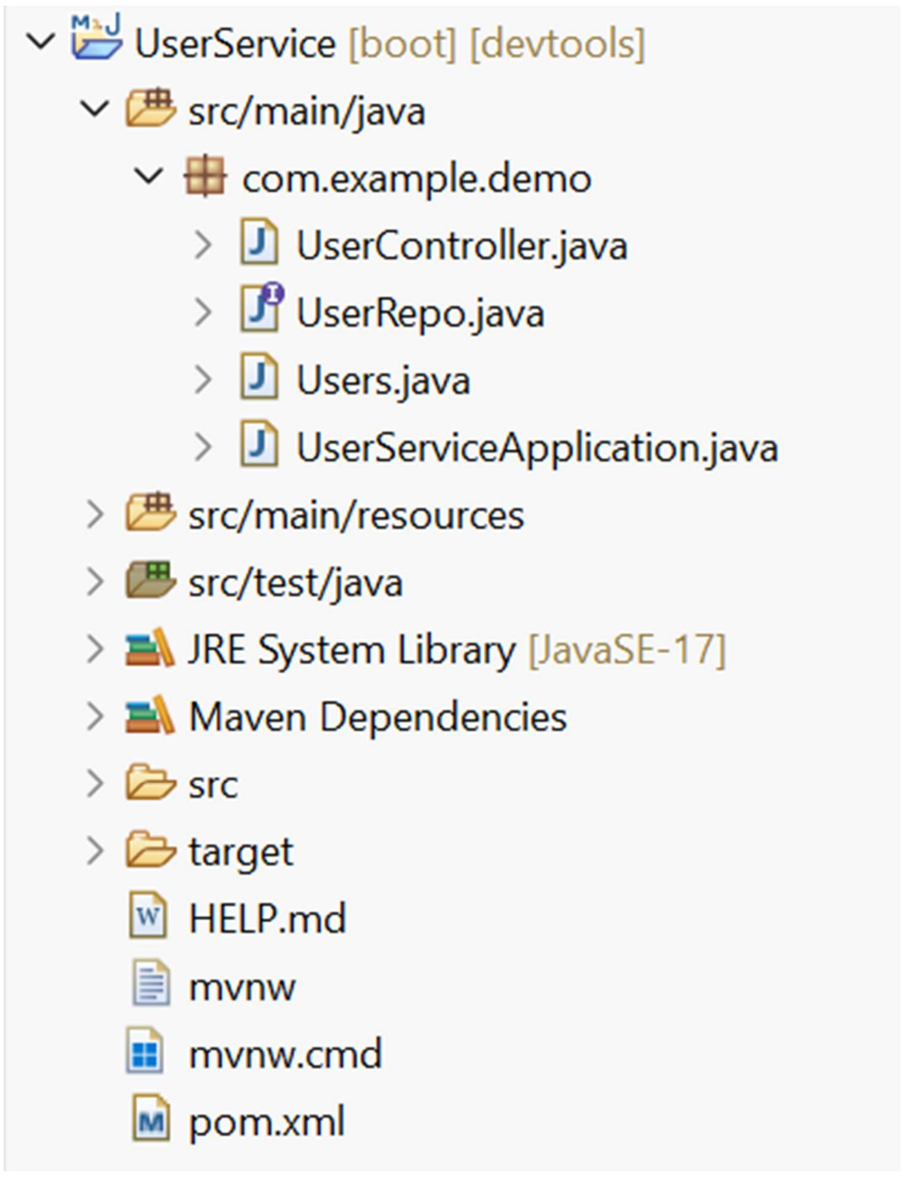
Difference Between Microservices and Monolithic Architecture

Aspect	Microservices Architecture	Monolithic Architecture
Structure	Small, independent services	Single large application
Team Structure	Small, independent teams	Centralized large team
Scalability	Scale services individually	Scale entire application
Deployment	Independent deployments	Single deployment
Resource Usage	Efficient, service-based	Resource usage for whole app
Development Speed	Faster development & updates	Slower due to large codebase
Flexibility	Multiple technologies allowed	Single technology stack
Maintenance	Easy to maintain small services	Difficult for large systems

Step-by-Step Microservices Implementation:-

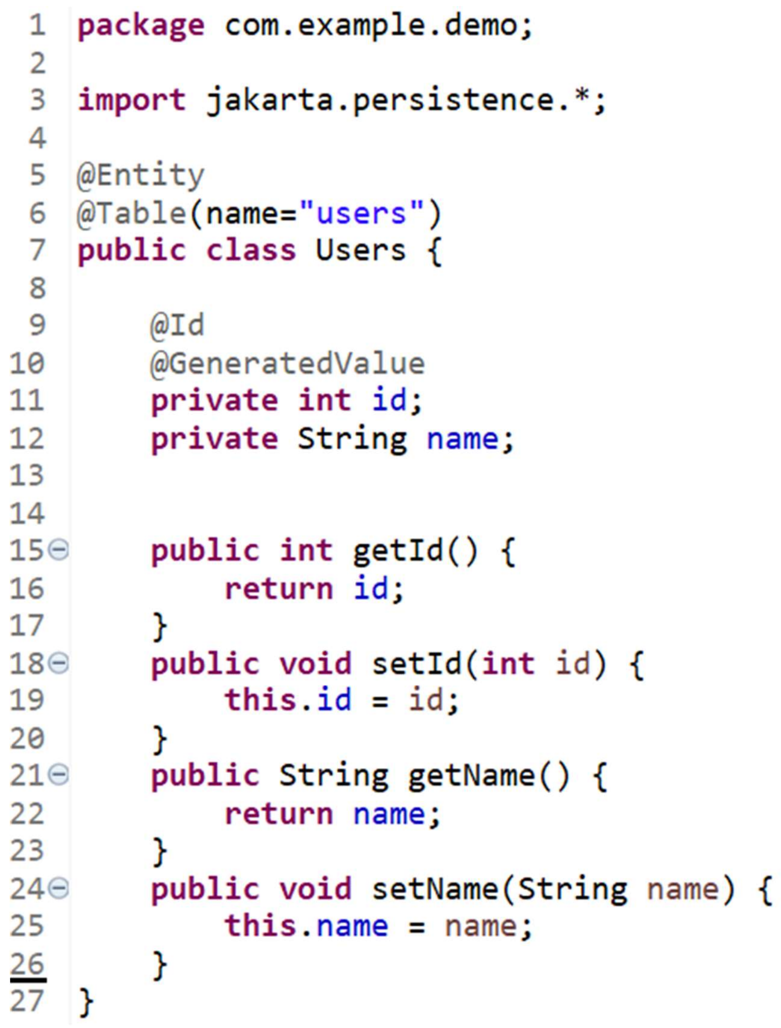
1. UserService – User Management Microservice –

Step 1 – Project Structure

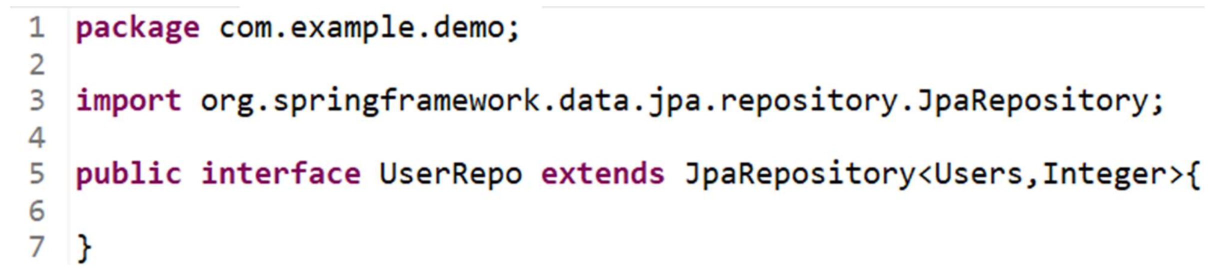


```
UserService [boot] [devtools]
  src/main/java
    com.example.demo
      UserController.java
      UserRepo.java
      Users.java
      UserServiceApplication.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

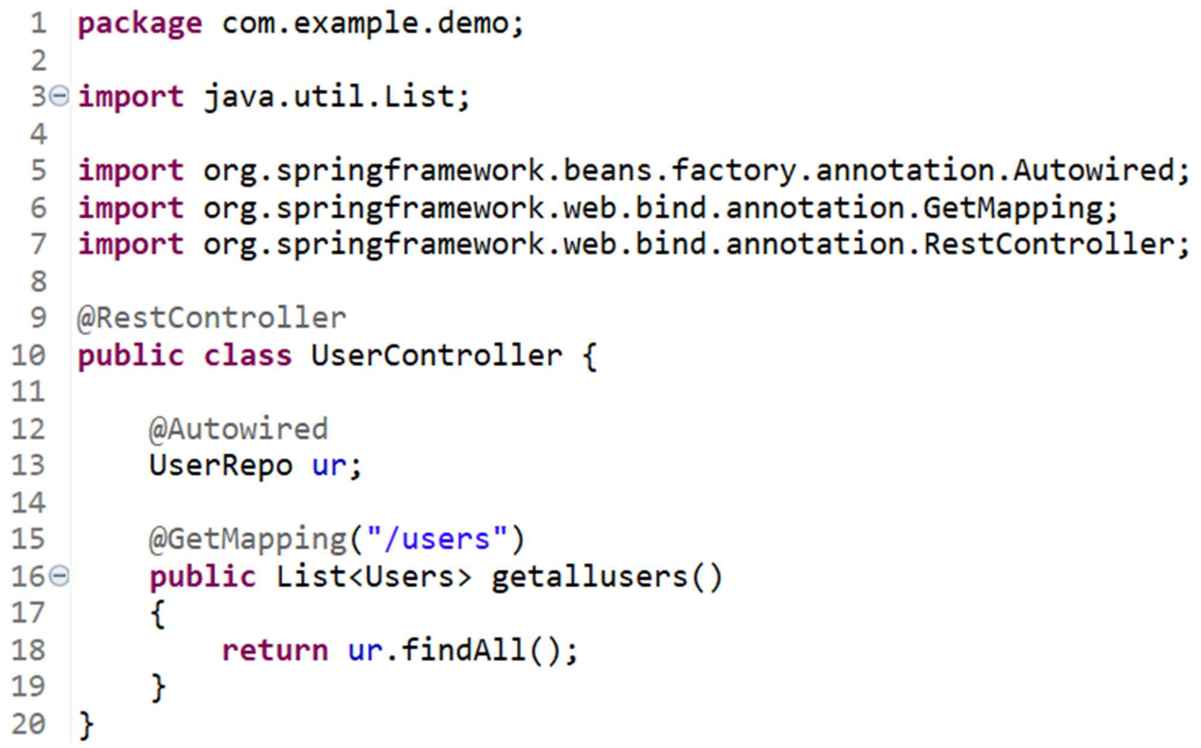
Step 2 - Create Entity Class :- Users.java


```java
package com.example.demo;

import jakarta.persistence.*;

@Entity
@Table(name="users")
public class Users {

    @Id
    @GeneratedValue
    private int id;
    private String name;


    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Step 3 - Create Repository Interface :- UserRepo.java

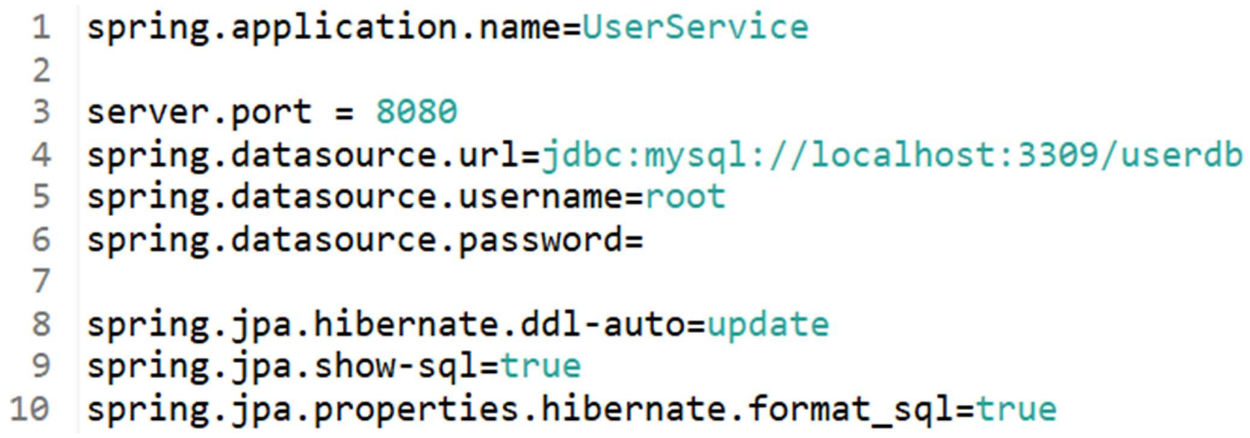

```java
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepo extends JpaRepository<Users,Integer>{

}
```

Step 4 - Create Controller :- UserController.java



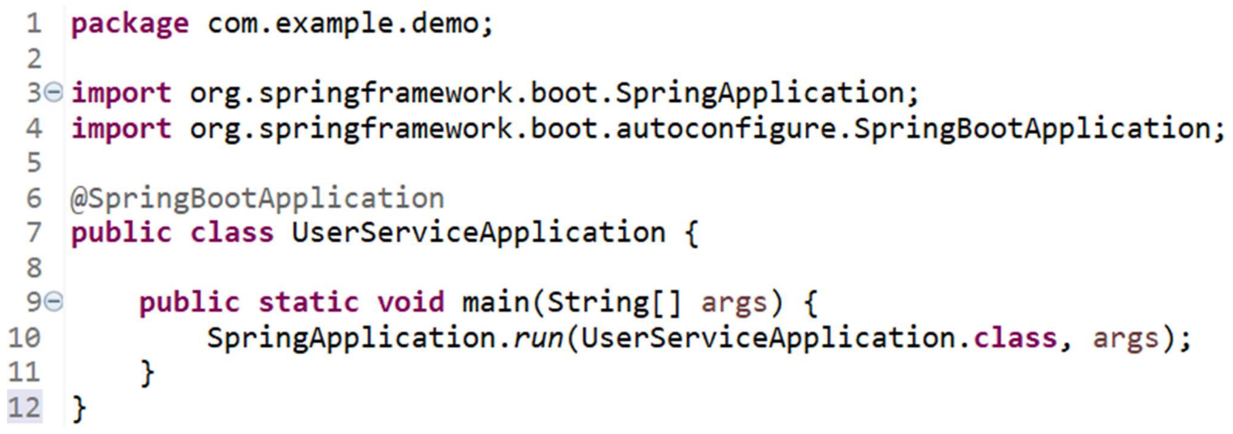
```java
package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    @Autowired
    UserRepo ur;

    @GetMapping("/users")
    public List<Users> getallusers()
    {
        return ur.findAll();
    }
}
```

Step 5 - Configure Application :- application.properties



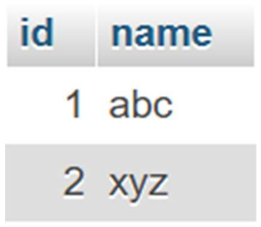
```properties
spring.application.name=UserService

server.port = 8080
spring.datasource.url=jdbc:mysql://localhost:3309/userdb
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Step 6 - Run UserService :- UserServiceApplication.java



```java
package com.example.demo;

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

@SpringBootApplication
public class UserServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(UserServiceApplication.class, args);
    }
}
```

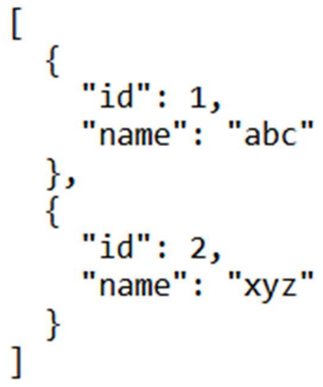
Step 7 – Database :- MySql



id	name
1	abc
2	xyz

Step 8 – Run on Server

http://localhost:8080/users

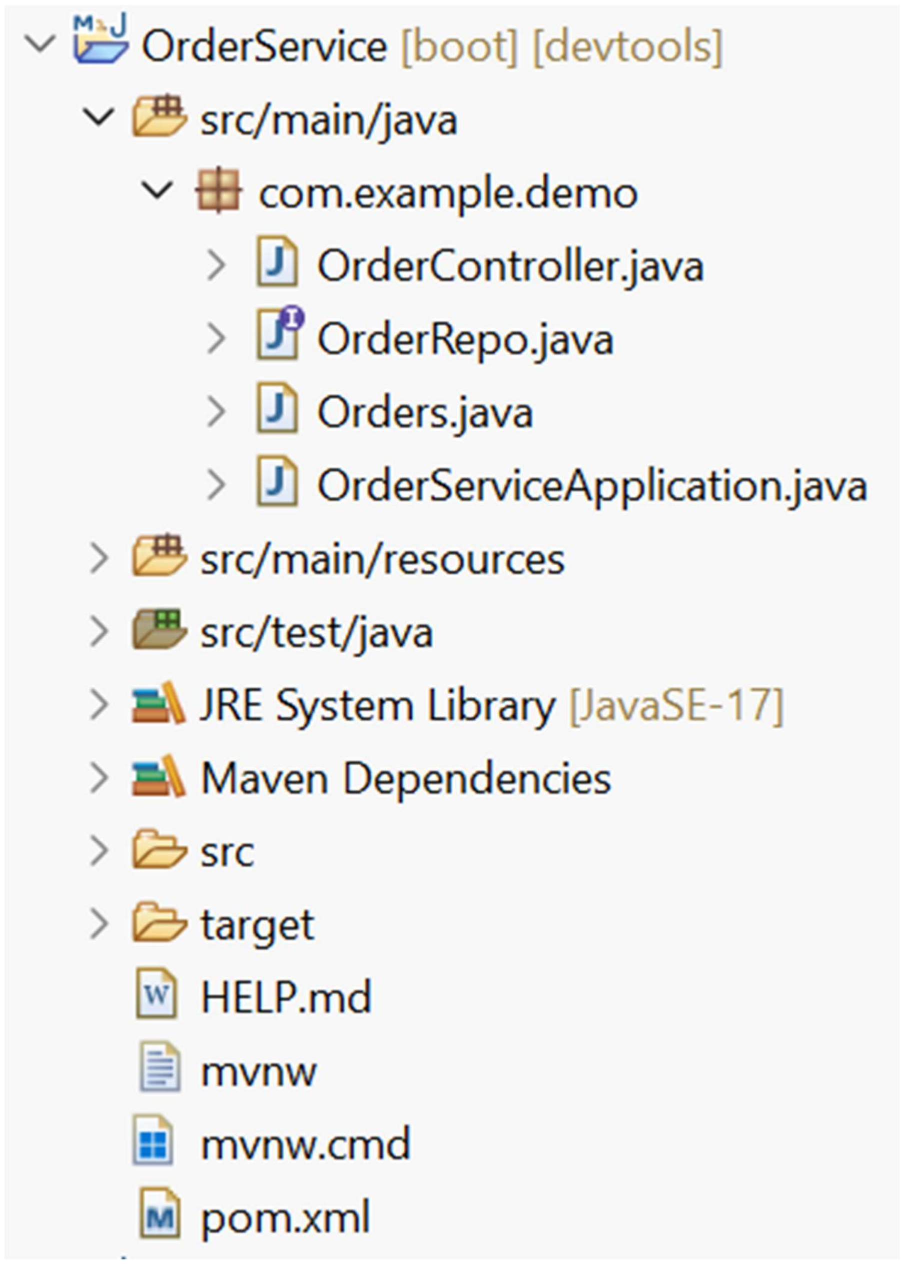


```
[
  {
    "id": 1,
    "name": "abc"
  },
  {
    "id": 2,
    "name": "xyz"
  }
]
```

2. OrderService – Order Management Microservice

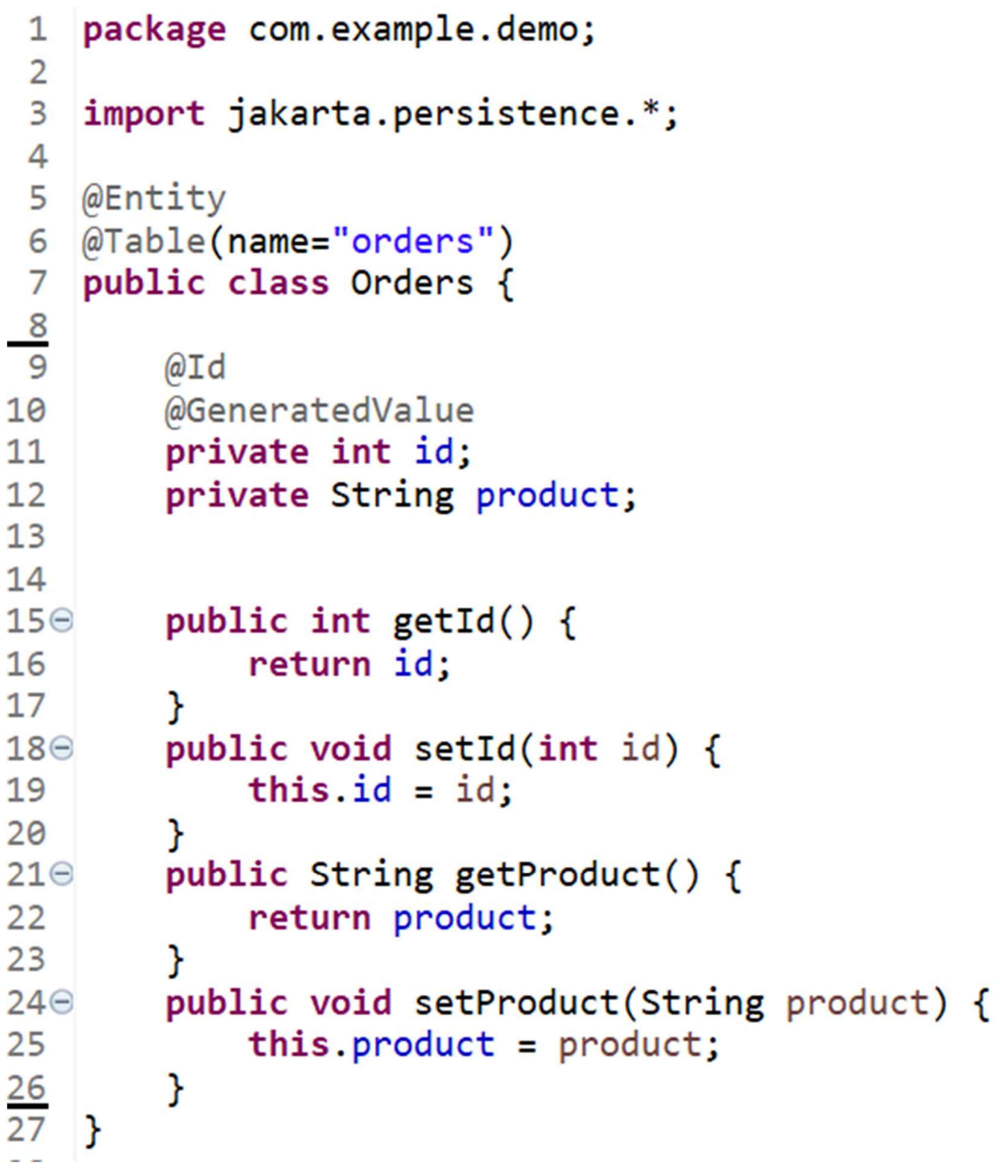
Step 1 – Project Structure

OrderService -



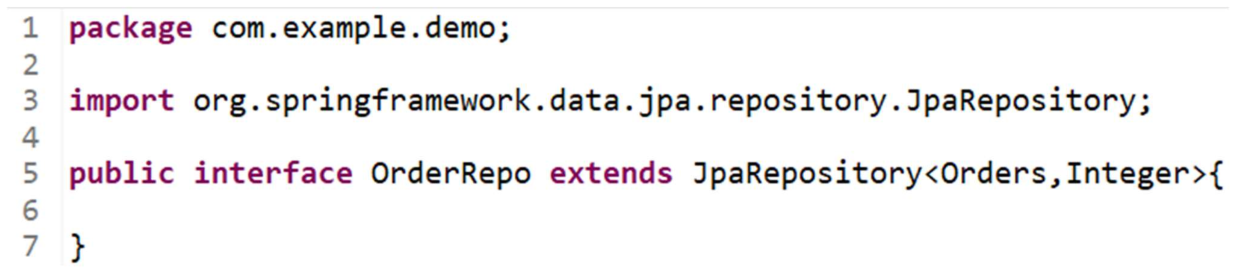
```
OrderService [boot] [devtools]
  src/main/java
    com.example.demo
      OrderController.java
      OrderRepo.java
      Orders.java
      OrderServiceApplication.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Step 2 - Create Entity Class :- Order.java



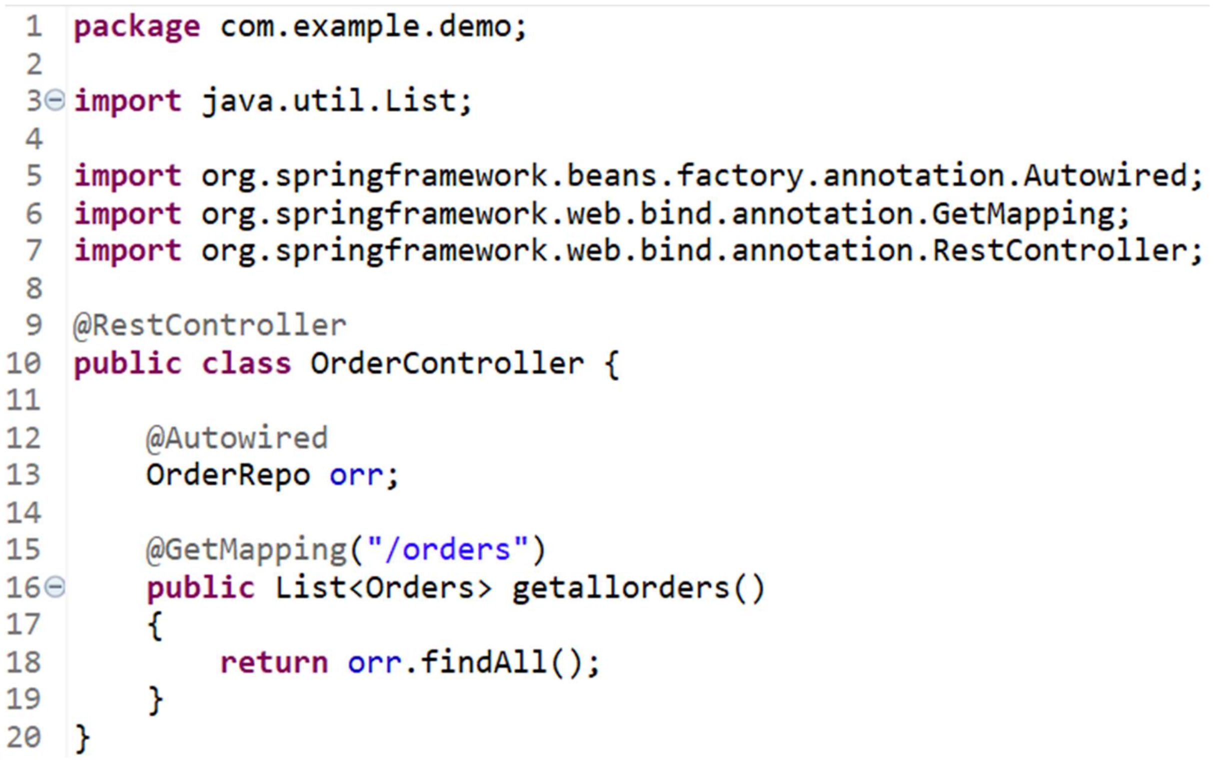
```java
package com.example.demo;

import jakarta.persistence.*;

@Entity
@Table(name="orders")
public class Orders {

    @Id
    @GeneratedValue
    private int id;
    private String product;


    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getProduct() {
        return product;
    }
    public void setProduct(String product) {
        this.product = product;
    }
}
```

Step 3 - Create Repository Interface :- OrderRepo.java



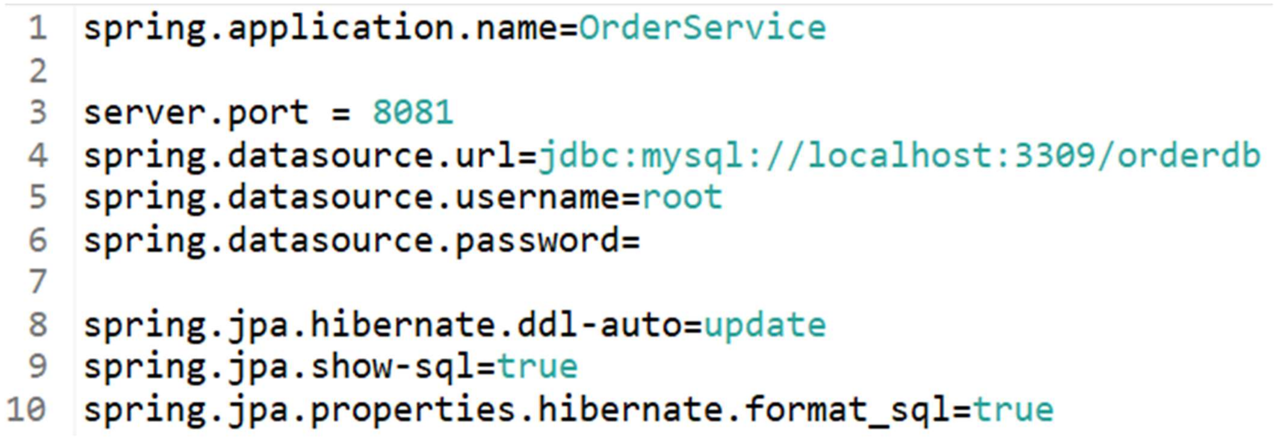
```java
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderRepo extends JpaRepository<Orders,Integer>{

}
```

Step 4 - Create Controller :- OrderController.java



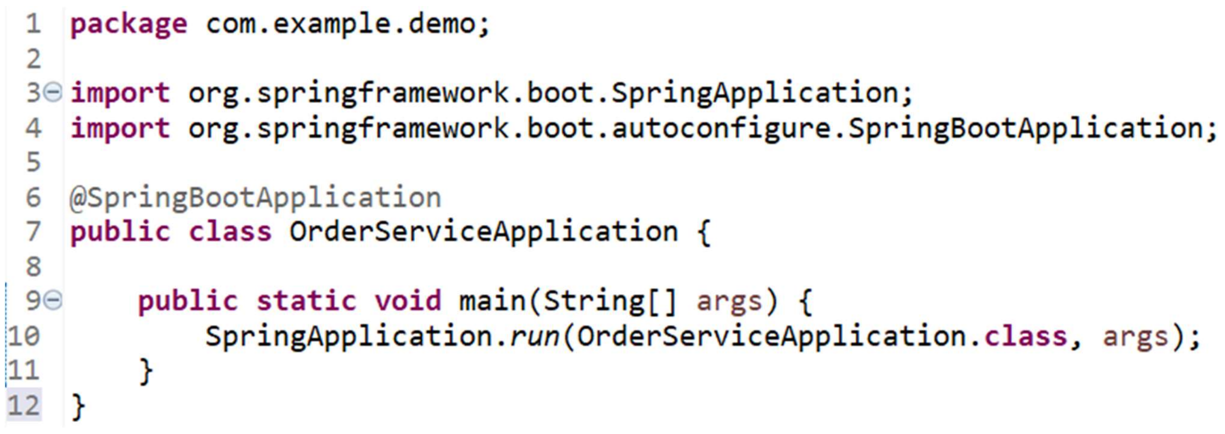
```java
package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class OrderController {

    @Autowired
    OrderRepo orr;

    @GetMapping("/orders")
    public List<Orders> getallorders()
    {
        return orr.findAll();
    }
}
```

Step 5 - Configure Application :- application.properties



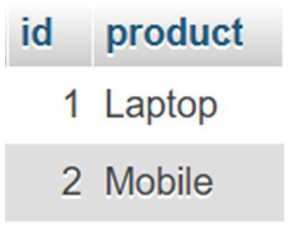
```properties
spring.application.name=OrderService

server.port = 8081
spring.datasource.url=jdbc:mysql://localhost:3309/orderdb
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Step 6 - Run OrderService :- OrderServiceApplication.java



```java
package com.example.demo;

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

@SpringBootApplication
public class OrderServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(OrderServiceApplication.class, args);
    }
}
```

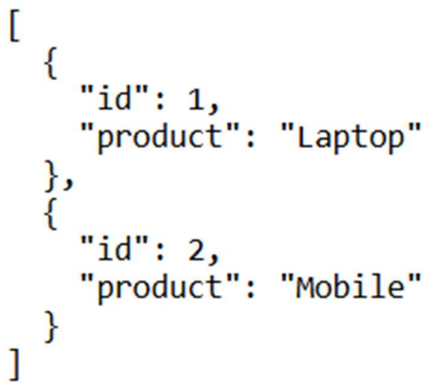
Step 7 – Database :- MySql



id	product
1	Laptop
2	Mobile

Step 8 – Run on Server

http://localhost:8081/orders

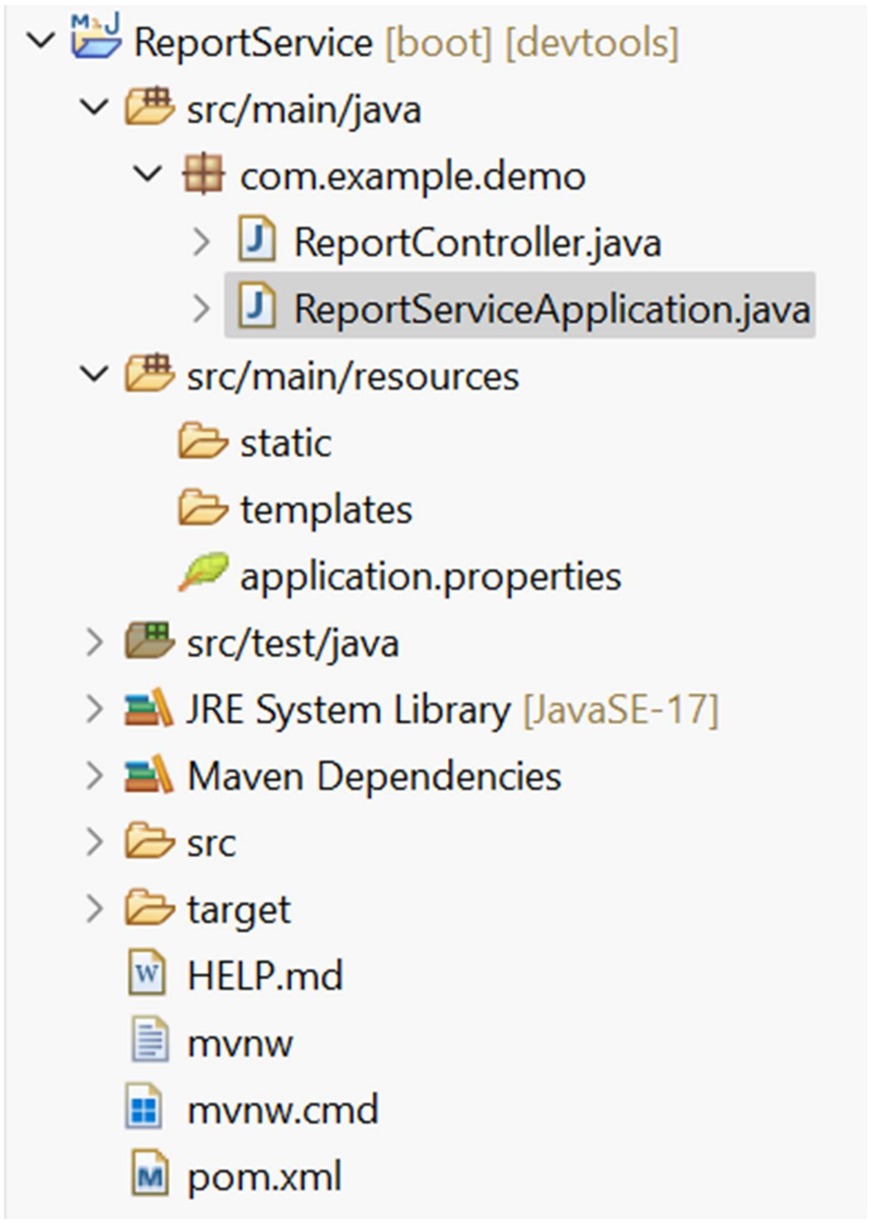


```
[
  {
    "id": 1,
    "product": "Laptop"
  },
  {
    "id": 2,
    "product": "Mobile"
  }
]
```

3 . ReportService – Report Generation Microservice

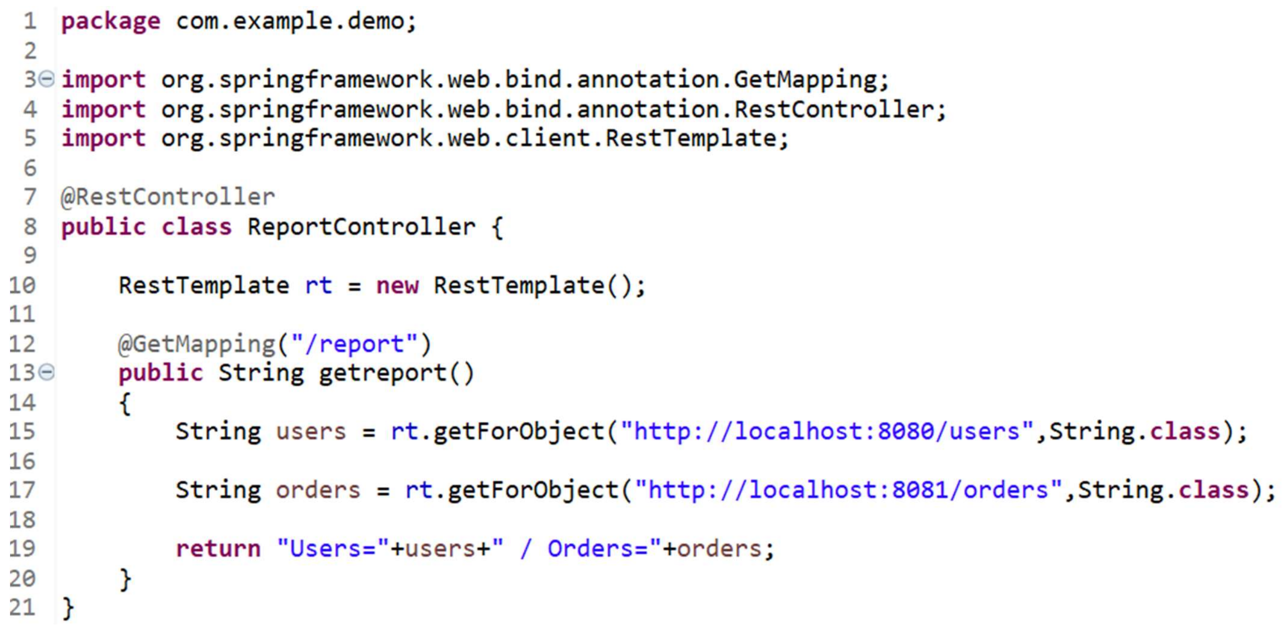
Step 1 – Project Structure

ReportService –



```
ReportService [boot] [devtools]
  src/main/java
    com.example.demo
      ReportController.java
      ReportServiceApplication.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Step 2 - Create Controller :- ReportController.java

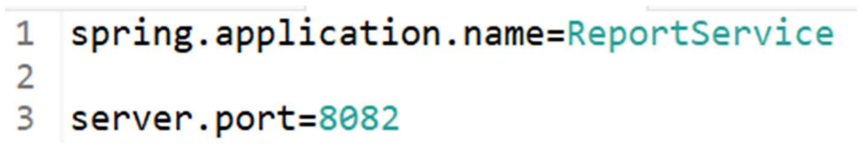


```java
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class ReportController {

    RestTemplate rt = new RestTemplate();

    @GetMapping("/report")
    public String getreport()
    {
        String users = rt.getForObject("http://localhost:8080/users",String.class);

        String orders = rt.getForObject("http://localhost:8081/orders",String.class);

        return "Users="+users+" / Orders="+orders;
    }
}
```

Step 3 - Call Other Microservices

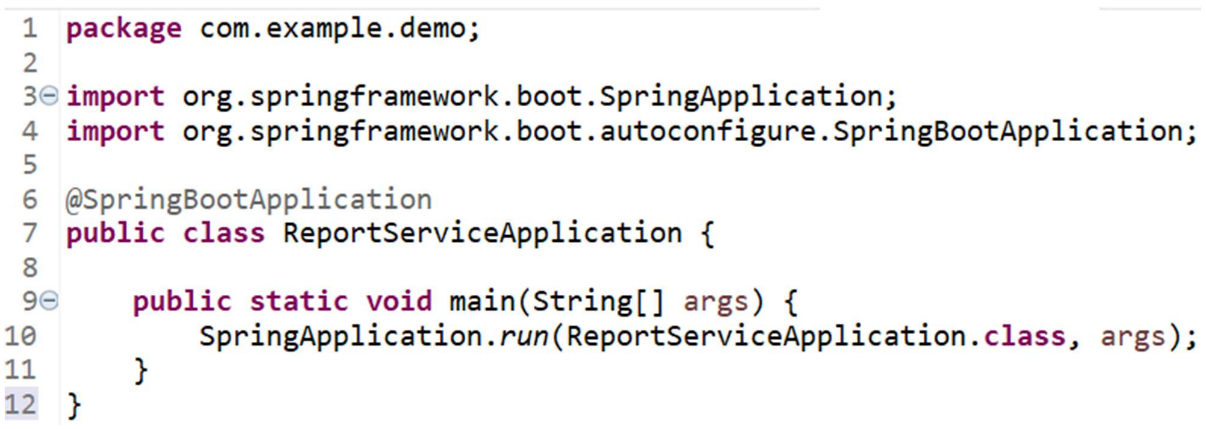
- **Use RestTemplate**
- **Fetch data from:**
  http://localhost:8080/users
  http://localhost:8081/orders

Step 4 - Configure Application :- application.properties



```properties
spring.application.name=ReportService

server.port=8082
```

Step 5 - Run ReportService :- ReportServiceApplication.java



```java
package com.example.demo;

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

@SpringBootApplication
public class ReportServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(ReportServiceApplication.class, args);
    }
}
```

Step 6 - Run on Server

http://localhost:8080/users

```
[{"id":1,"name":"abc"},{"id":2,"name":"xyz"}]
```

http://localhost:8081/orders

```
[{"id":1,"product":"Laptop"},{"id":2,"product":"Mobile"}]
```

http://localhost:8082/report

```
Users=[{"id":1,"name":"abc"},{"id":2,"name":"xyz"}] / Orders=[{"id":1,"product":"Laptop"},{"id":2,"product":"Mobile"}]
```

How All Services Work Together

```
Client
  ↓
ReportService
  ↓           ↓
UserService  OrderService
```