**Digital Nurture 4.0 – Week 4**

Spring REST using Spring Boot 3

ADDDITIONAL EXERCISES

Filename: 3. spring-rest-handson

**EXERCISE 1: Problem Statement - Display Employee List and Edit Employee form using RESTful Web Service**

**PART 1: SPRING BOOT BACKEND (REST + XML)**

Step 1: Create Spring Boot Project in IntelliJ

1. Open IntelliJ → Click **New Project**
2. Select **Spring Initializr**
3. Fill:
   * Group: com.example
   * Artifact: employeeapp
4. Select Java 17 or above
5. Select dependencies:
   * Spring Web
   * Spring Context
6. Click **Next** → Finish

**Step 2: Create Employee Model Class**

1. Right-click on: src/main/java/com/example/employeeapp
2. Create folder: model
3. Inside it, create class: Employee.java

package com.example.employeeapp.model;

public class Employee {

private int id;

private String name;

private String designation;

private double salary;

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; }

public String getDesignation() { return designation; }

public void setDesignation(String designation) { this.designation = designation; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

}

**Step 3: Create employees.xml**

1. Right-click src/main/resources → New → File → employees.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="employee1" class="com.example.employeeapp.model.Employee">

<property name="id" value="1"/>

<property name="name" value="John Doe"/>

<property name="designation" value="Developer"/>

<property name="salary" value="50000"/>

</bean>

<bean id="employee2" class="com.example.employeeapp.model.Employee">

<property name="id" value="2"/>

<property name="name" value="Jane Smith"/>

<property name="designation" value="Manager"/>

<property name="salary" value="75000"/>

</bean>

<bean id="employeeList" class="java.util.ArrayList">

<constructor-arg>

<list>

<ref bean="employee1"/>

<ref bean="employee2"/>

</list>

</constructor-arg>

</bean>

</beans>

**Step 4: Create Employee REST Controller**

1. Create a folder: controller
2. Inside, create EmployeeController.java

package com.example.employeeapp.controller;

import com.example.employeeapp.model.Employee;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

import java.util.List;

public class EmployeeController {

private List<Employee> employeeList;

public EmployeeController() {

ApplicationContext context = new ClassPathXmlApplicationContext("employees.xml");

this.employeeList = (List<Employee>) context.getBean("employeeList");

}

public List<Employee> getAllEmployees() {

return employeeList;

}

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable int id) {

return employeeList.stream().filter(e -> e.getId() == id).findFirst().orElse(null);

}

}

**Step 5: Main Application Class**

Make sure EmployeeappApplication.java

package com.example.employeeapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

public class EmployeeappApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeappApplication.class, args);

}

}

**Step 6: Add Dependencies in pom.xml**

Paste this in <dependencies>:

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>jakarta.xml.bind</groupId>

<artifactId>jakarta.xml.bind-api</artifactId>

</dependency>

**Step 7: Run the App**

Right-click EmployeeappApplication.java → Click **Run**

**PART 2: ANGULAR FRONTEND**

**Step 1: Create Angular App**

Open terminal:

ng new employee-ui

cd employee-ui

Step 2: Generate Components and Service

ng generate component employee-list

ng generate component employee-edit

ng generate service employee

**Step 3: Create Employee Model**

In src/app/employee.model.ts:

export interface Employee {

id: number;

name: string;

designation: string;

salary: number;

}

Step 4: Setup REST Service (employee.service.ts)

@Injectable({ providedIn: 'root' })

export class EmployeeService {

private baseUrl = 'http://localhost:8080/api/employees';

constructor(private http: HttpClient) {}

getEmployees(): Observable<Employee[]> {

return this.http.get<Employee[]>(this.baseUrl);

}

getEmployeeById(id: number): Observable<Employee> {

return this.http.get<Employee>(`${this.baseUrl}/${id}`);

}

}

Step 5: Show List (employee-list.component.ts)

export class EmployeeListComponent implements OnInit {

employees: Employee[] = [];

constructor(private employeeService: EmployeeService, private router: Router) {}

ngOnInit(): void {

this.employeeService.getEmployees().subscribe(data => {

this.employees = data;

});

}

editEmployee(id: number) {

this.router.navigate(['/edit', id]);

}

}

HTML:

<table>

<tr \*ngFor="let emp of employees">

<td>{{ emp.name }}</td>

<td>{{ emp.designation }}</td>

<td>{{ emp.salary }}</td>

<td><button (click)="editEmployee(emp.id)">Edit</button></td>

</tr>

</table>

Step 6: Edit Form (employee-edit.component.ts)

export class EmployeeEditComponent implements OnInit {

employee: Employee = { id: 0, name: '', designation: '', salary: 0 };

constructor(private service: EmployeeService, private route: ActivatedRoute) {}

ngOnInit(): void {

const id = +this.route.snapshot.paramMap.get('id')!;

this.service.getEmployeeById(id).subscribe(data => this.employee = data);

}

}

HTML:

<form \*ngIf="employee">

<input [(ngModel)]="employee.name" name="name">

<input [(ngModel)]="employee.designation" name="designation">

<input [(ngModel)]="employee.salary" name="salary">

<button>Save</button>

</form>

**Step 7: Add Routing**

In app-routing.module.ts:

const routes: Routes = [

{ path: '', component: EmployeeListComponent },

{ path: 'edit/:id', component: EmployeeEditComponent },

];

**Step 8: Enable HttpClient**

In app.module.ts:

import { HttpClientModule } from '@angular/common/http';

@NgModule({

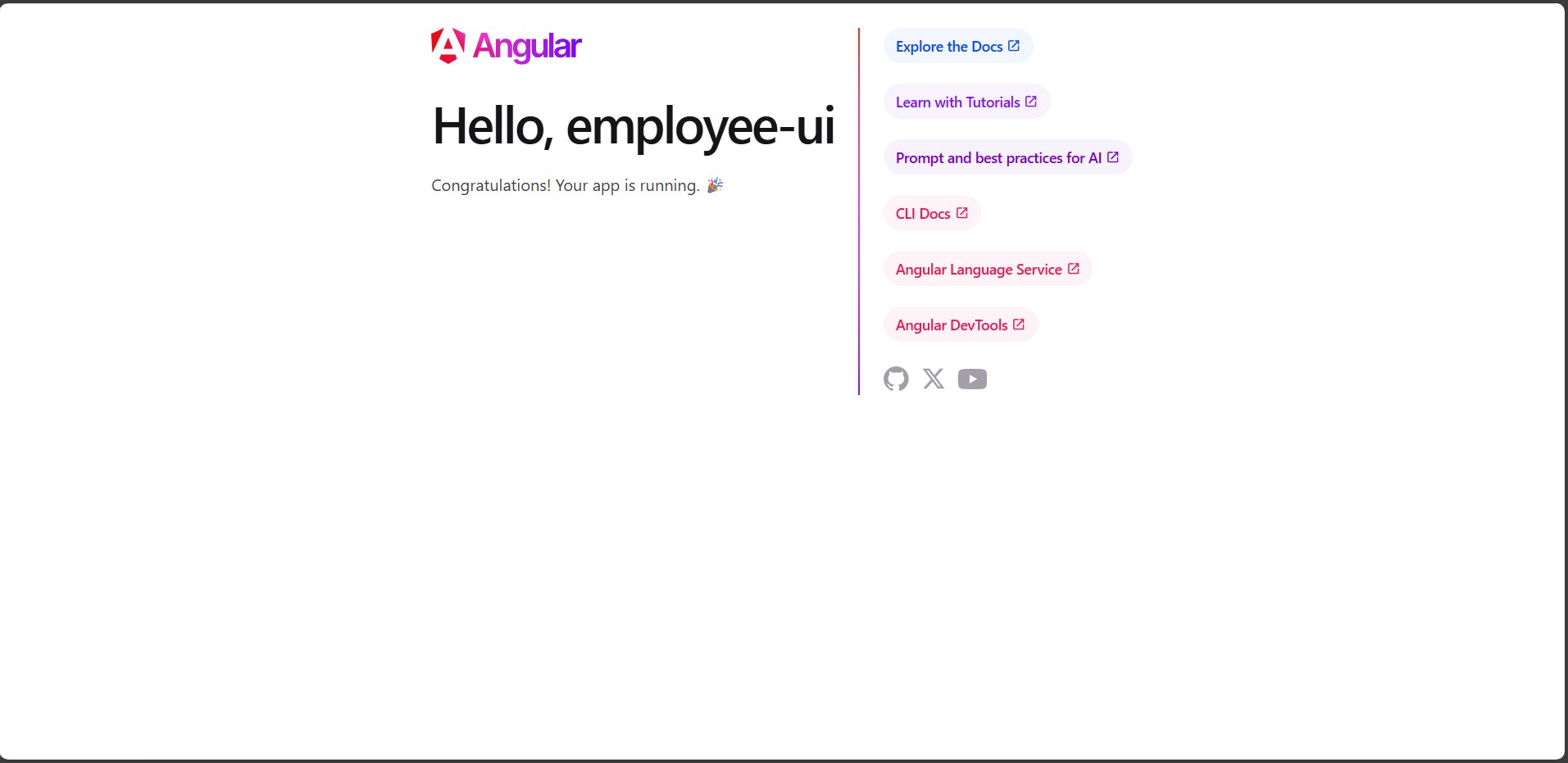
imports: [BrowserModule, HttpClientModule, FormsModule, AppRoutingModule],

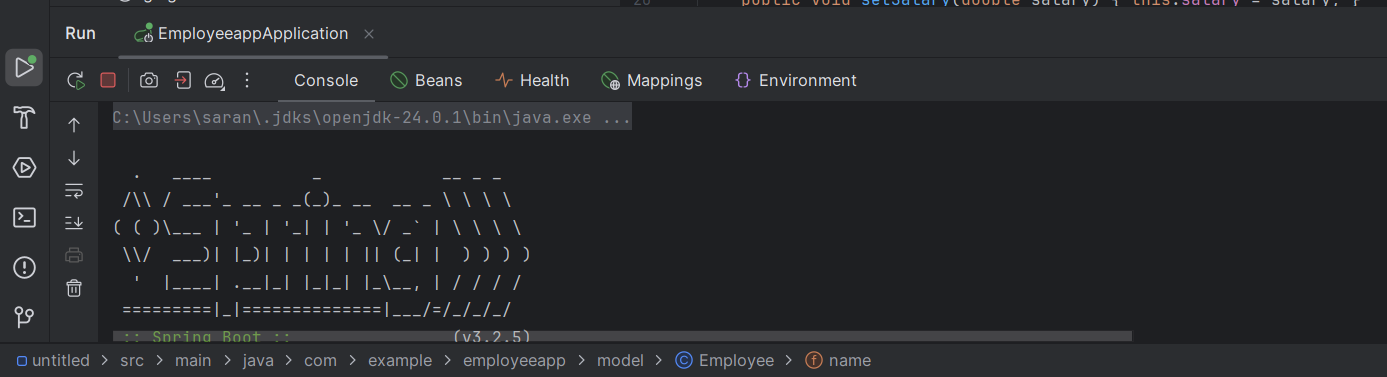
})

Step 9: Run Angular App

ng serve

**OUTPUT:**

****



**EXERCISE 2: Create static employee list data using spring xml configuration**

**Step 1: Create New Maven Project**

1. **Open IntelliJ IDEA → File → New → Project**
2. Select **Maven**, click **Next**
3. GroupId: com.example  
   ArtifactId: employee-xml
4. Click **Finish**

**Step 2: Create Package Structure**

Right-click on src/main/java, create packages:

com.example.model

com.example.dao

com.example

**Step 3: Add Spring Dependencies in pom.xml**

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

``>

Step 4: Create Model Classes

`com.example.model`

package com.example.model;

public class Skill {

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; }

}

**Department.java** – inside com.example.model:

package com.example.model;

public class Department {

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; }

}

**Employee.java** – inside com.example.model:

package com.example.model;

import java.util.List;

public class Employee {

private int id;

private String name;

private double salary;

private boolean permanent;

private Department department;

private List<Skill> skills;

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; }

public double getSalary() { return salary; }

public void setSalary(double salary) { this.salary = salary; }

public boolean isPermanent() { return permanent; }

public void setPermanent(boolean permanent) { this.permanent = permanent; }

public Department getDepartment() { return department; }

public void setDepartment(Department department) { this.department = department; }

public List<Skill> getSkills() { return skills; }

public void setSkills(List<Skill> skills) { this.skills = skills; }

}

**Step 5: Create EmployeeDao.java**

Inside com.example.dao

package com.example.dao;

import com.example.model.Employee;

import java.util.List;

public class EmployeeDao {

private static List<Employee> EMPLOYEE\_LIST;

public EmployeeDao(List<Employee> employeeList) {

EMPLOYEE\_LIST = employeeList;

}

public List<Employee> getAllEmployees() {

return EMPLOYEE\_LIST;

}

}

**Step 6: Create employee.xml inside src/main/resources**

Right-click on resources → New → XML File → employee.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

<http://www.springframework.org/schema/beans/spring-beans.xsd>">

Step 7: Create Main.java inside com.example:

package com.example;

import com.example.dao.EmployeeDao;

import com.example.model.Employee;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("employee.xml");

EmployeeDao dao = context.getBean("employeeDao", EmployeeDao.class);

for (Employee emp : dao.getAllEmployees()) {

System.out.println(emp.getName() + " - " + emp.getDepartment().getName());

}

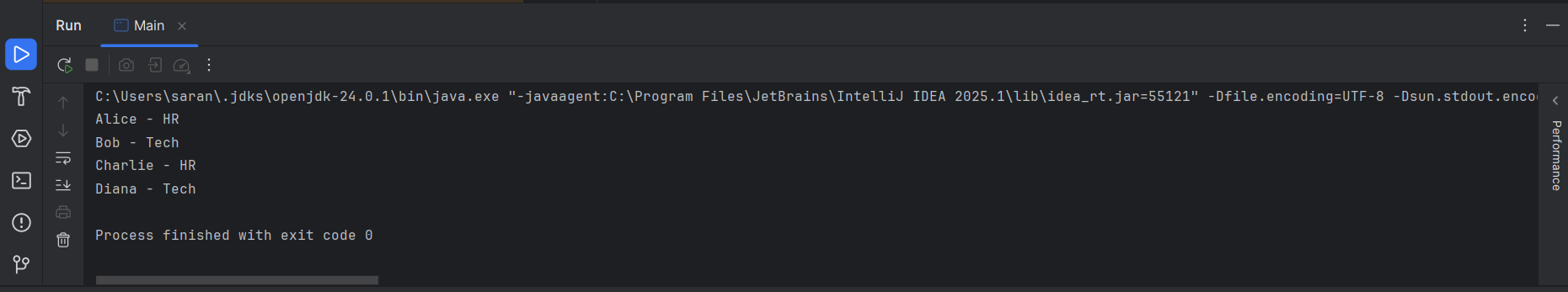
}

}

**Step 8: Run the Project**

Right-click Main.java → **Run 'Main.main()'**

OUTPUT:



**EXERCISE 3: Create REST service to gets all employees**

**Step 1: Create EmployeeService.java**

src/main/java/com/example/service/EmployeeService.java

package com.example.service;

import com.example.dao.EmployeeDao;

import com.example.model.Employee;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeDao employeeDao;

@Transactional

public List<Employee> getAllEmployees() {

return employeeDao.getAllEmployees();

}

}

**Step 2: Create EmployeeController.java**

src/main/java/com/example/controller/EmployeeController.java

package com.example.controller;

import com.example.model.Employee;

import com.example.service.EmployeeService;

import org.springframework.beans.factory.annotation.Autowired;

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

import java.util.List;

@RestController

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping("/employees")

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

}

**Step 3: Update dispatcher-servlet.xml**

src/main/webapp/WEB-INF/dispatcher-servlet.xml

<context:component-scan base-package="com.example" />

<tx:annotation-driven />

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/>

<bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"/>

<import resource="classpath:employee.xml"/>

**Step 4: Update web.xml**

src/main/webapp/WEB-INF/web.xml

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

version="3.1">

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

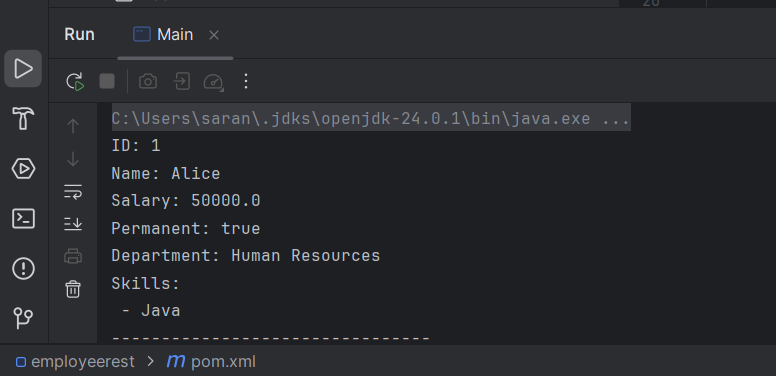
</web-app>

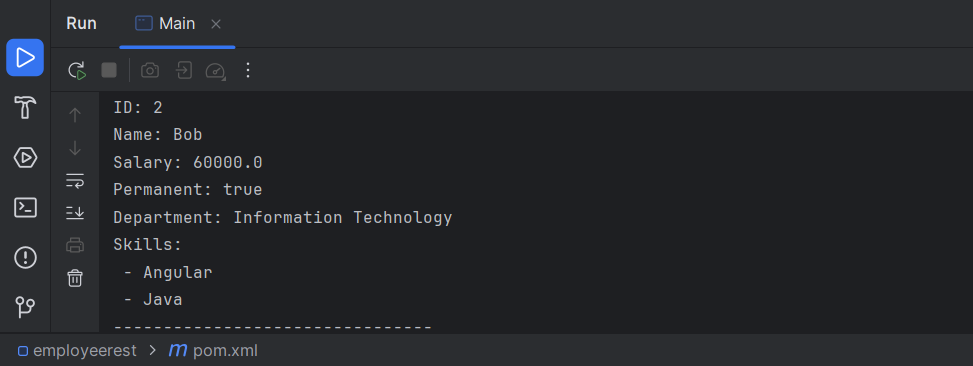
**Step 5: Run the Application on Tomcat**

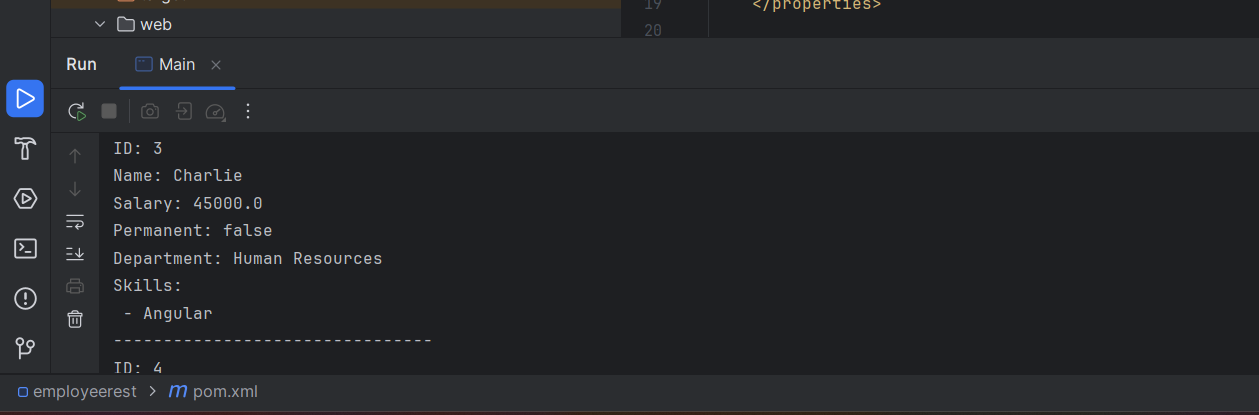
**Set Up Tomcat in IntelliJ**

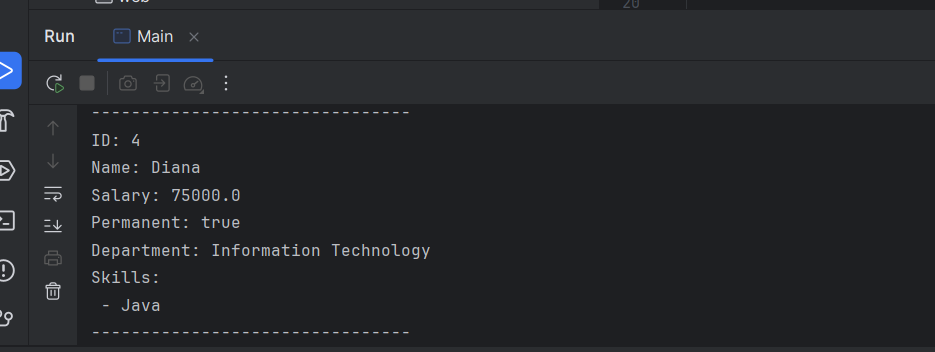
1. **File → Project Structure → Artifacts**
   * Add Artifact: Web Application: Exploded → Name: employee-rest:war exploded
2. **Run → Edit Configurations**
   * Add new Tomcat Server → Local
   * Deployment tab → + → Add your artifact
3. Click Apply and Run

**Output:**









**EXERCISE 4: Create REST service for department**

Step 1: Create Maven Project in IntelliJ

1. Open IntelliJ IDEA
2. Click File → New → Project
3. Choose Maven
4. Check Create from archetype → Select maven-archetype-webapp
5. Name it employeerest
6. Set project location (ex: C:\Users\saran\IdeaProjects\employeerest)
7. Finish → Wait for Maven to load

**Step 2: Add Folder Structure**

src/

└── main/

├── java/

│ └── com/

│ └── example/

│ ├── controller/

│ ├── service/

│ ├── dao/

│ └── model/

├── resources/

│ └── employee.xml

└── webapp/

└── WEB-INF/

├── dispatcher-servlet.xml

└── web.xml

**Step 3: Add pom.xml Dependencies**

Replace the contents of pom.xml with:

<project xmlns="http://maven.apache.org/POM/4.0.0" ...>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>employeerest</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.13.5</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<finalName>employeerest</finalName>

</build>

</project>

**Step 4: Add Spring XML Files**

**web.xml (WEB-INF/web.xml)**

<web-app ...>

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

**dispatcher-servlet.xml (WEB-INF/dispatcher-servlet.xml)**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd">

<context:component-scan base-package="com.example"/>

<mvc:annotation-driven/>

<import resource="classpath:employee.xml"/>

</beans>

**Step 5: Add Java Classes**

**Model Classes (Employee.java, Department.java, Skill.java)**

Each with:

* fields (id, name, etc.)
* getters and setters  
  Let me know if you want the exact code — I’ll paste it.

**DAO**

* EmployeeDao.java → uses constructor injection for employee list
* DepartmentDao.java → uses constructor injection for department list

**Service**

* EmployeeService.java → injects EmployeeDao, returns list
* DepartmentService.java → injects DepartmentDao, returns list

**Controller**

* EmployeeController.java → /employees
* DepartmentController.java → /departments

**Step 6: Add and Configure employee.xml**

Paste the **updated employee.xml** from the previous message  
Save in src/main/resources

**Step 7: Build and Run the Project**

1. Right-click project → **Add Framework Support** → select Web
2. Go to **Edit Configurations**:
   * Add **Tomcat Server → Local**
   * Deploy artifact: employeerest:war exploded
3. Run the project

**Output:**

