simple example of a **Spring Web REST API** for an Employee case study without using Spring Boot, and it uses basic collections for storage (like ArrayList) instead of a database.

**1. Maven Project Setup (pom.xml)**

First, make sure to include the necessary dependencies in your pom.xml for Spring Web and the required annotations:

xml

Copy code

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring Web (for REST API) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Servlet API -->

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

<!-- Jackson for JSON Serialization -->

<dependency>

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

<artifactId>jackson-databind</artifactId>

<version>2.14.0</version>

</dependency>

</dependencies>

**2. Employee Model Class**

This is the simple Employee class that represents an employee:

java

Copy code

public class Employee {

private int id;

private String name;

private String position;

// Constructor

public Employee(int id, String name, String position) {

this.id = id;

this.name = name;

this.position = position;

}

// Getters and Setters

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 getPosition() {

return position;

}

public void setPosition(String position) {

this.position = position;

}

}

**3. EmployeeService**

This class holds the collection of employees and manages CRUD operations. We’ll use an ArrayList to store the employees.

java

Copy code

import java.util.ArrayList;

import java.util.List;

public class EmployeeService {

private List<Employee> employeeList = new ArrayList<>();

public EmployeeService() {

// Sample employees

employeeList.add(new Employee(1, "John Doe", "Developer"));

employeeList.add(new Employee(2, "Jane Smith", "Manager"));

}

// Get all employees

public List<Employee> getAllEmployees() {

return employeeList;

}

// Get employee by ID

public Employee getEmployeeById(int id) {

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

}

// Add employee

public void addEmployee(Employee employee) {

employeeList.add(employee);

}

// Delete employee by ID

public boolean deleteEmployeeById(int id) {

return employeeList.removeIf(e -> e.getId() == id);

}

}

**4. EmployeeController (REST Controller)**

Here’s the REST controller that handles API requests for Employee. It will return JSON responses.

java

Copy code

import org.springframework.stereotype.Controller;

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

import java.util.List;

@Controller

@RequestMapping("/employees")

public class EmployeeController {

private EmployeeService employeeService = new EmployeeService();

// Get all employees

@GetMapping

@ResponseBody

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

// Get employee by ID

@GetMapping("/{id}")

@ResponseBody

public Employee getEmployeeById(@PathVariable int id) {

return employeeService.getEmployeeById(id);

}

// Add a new employee

@PostMapping

@ResponseBody

public String addEmployee(@RequestBody Employee employee) {

employeeService.addEmployee(employee);

return "Employee added successfully!";

}

// Delete employee by ID

@DeleteMapping("/{id}")

@ResponseBody

public String deleteEmployee(@PathVariable int id) {

boolean isRemoved = employeeService.deleteEmployeeById(id);

if (isRemoved) {

return "Employee deleted successfully!";

} else {

return "Employee not found!";

}

}

}

**5. Web.xml (For Dispatcher Servlet Configuration)**

Since this is not using Spring Boot, you’ll need to configure the DispatcherServlet in web.xml.

xml

Copy code

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

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

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

version="3.1">

<display-name>Employee REST API</display-name>

<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/spring-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>

**6. Spring Configuration (spring-servlet.xml)**

Define Spring's configuration in spring-servlet.xml:

xml

Copy code

<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="viewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp" />

</bean>

<!-- Enable annotation-based controllers -->

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

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

<!-- Define controller scanning -->

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

</beans>

**7. Testing the API**

You can test the REST API using tools like **Postman** or **curl**.

* **GET** /employees – Retrieve all employees.
* **GET** /employees/{id} – Retrieve a specific employee.
* **POST** /employees – Add a new employee (send JSON body).
* **DELETE** /employees/{id} – Delete an employee by ID.

For example, adding a new employee via **POST** request:

json

Copy code

{

"id": 3,

"name": "Sara Connor",

"position": "Designer"

}

This example creates a basic REST API without Spring Boot, using simple collections for employee management. Let me know if you need further assistance!