**Spring Boot Employee Management Application Setup Guide**

This guide explains how to create a Spring Boot application for managing employee records. The application will allow adding employee data via a POST request and listing employee data via a GET request. The records will be returned and received in JSON format.

**1. Prerequisites**

Before setting up the Spring Boot application, ensure the following tools are installed:

* **Java JDK** (version 11 or higher)
* **Maven** (for building the project)
* **Postman** (for testing API endpoints)

**2. Create Spring Boot Project**

**Option 1: Use Spring Initializr**

1. Go to [Spring Initializr](https://start.spring.io/).
2. Configure the project as follows:
   * **Project**: Maven Project
   * **Language**: Java
   * **Spring Boot Version**: Select the latest stable version
   * **Group**: com.example
   * **Artifact**: employee-management
   * **Packaging**: Jar
   * **Java Version**: 11 (or any version compatible with your system)
3. Add dependencies:
   * **Spring Web** (for RESTful web services)
   * **Spring Boot DevTools** (for automatic restarts during development)
4. Click **Generate** to download the project zip file. Extract it to your working directory.

**Option 2: Use VS Code’s Spring Boot Plugin**

Alternatively, you can use the Spring Boot plugin in Visual Studio Code to create the project.

**3. Project Structure**

Once the project is generated, it will have the following structure:

employee-management/  
│  
├── src/  
│ ├── main/  
│ │ ├── java/  
│ │ │ └── com/  
│ │ │ └── example/  
│ │ │ └── employeemanagement/  
│ │ │ ├── Employee.java  
│ │ │ ├── EmployeeController.java  
│ │ │ ├── EmployeeService.java  
│ │ │ └── EmployeeManagementApplication.java  
│ │ ├── resources/  
│ │ │ └── application.properties  
│ └── test/  
│ └── java/  
│ └── com/  
│ └── example/  
│ └── employeemanagement/  
│ └── EmployeeManagementApplicationTests.java  
└── pom.xml

**4. Define the Employee Model**

Create a Java class Employee.java under the com.example.employeemanagement package to represent an employee entity.

package com.example.employeemanagement;  
  
public class Employee {  
 private String id;  
 private String name;  
 private String position;  
 private String department;  
  
 *// Getters and setters*  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String 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;  
 }  
  
 public String getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(String department) {  
 this.department = department;  
 }  
}

**5. Create the EmployeeService**

The service will manage the business logic of adding and retrieving employees.

package com.example.employeemanagement;  
  
import org.springframework.stereotype.Service;  
import java.util.ArrayList;  
import java.util.List;  
  
@Service  
public class EmployeeService {  
  
 private List<Employee> employees = new ArrayList<>();  
  
 public List<Employee> getAllEmployees() {  
 return employees;  
 }  
  
 public void addEmployee(Employee employee) {  
 employees.add(employee);  
 }  
}

**6. Create the EmployeeController**

This controller will expose the RESTful endpoints for adding and listing employees.

package com.example.employeemanagement;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/employees")  
public class EmployeeController {  
  
 @Autowired  
 private EmployeeService employeeService;  
  
 *// Endpoint to get all employees*  
 @GetMapping  
 public List<Employee> getAllEmployees() {  
 return employeeService.getAllEmployees();  
 }  
  
 *// Endpoint to add a new employee*  
 @PostMapping  
 public void addEmployee(@RequestBody Employee employee) {  
 employeeService.addEmployee(employee);  
 }  
}

**7. Create the Main Application Class**

The EmployeeManagementApplication.java class is the entry point for your Spring Boot application.

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

**8. Configuration in application.properties**

You can configure the application’s settings, such as the server port, in the application.properties file located under src/main/resources.

*# application.properties*  
server.port=8080

**9. Add Maven Dependencies**

Ensure that your pom.xml file has the required dependencies. Below is an example configuration for Maven:

<dependencies>  
 *<!-- Spring Web dependency for REST API functionality -->*  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
   
 *<!-- Spring Boot DevTools for auto-restart during development -->*  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 *<!-- Spring Boot Test dependency for unit tests -->*  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

**10. Run the Application**

To run the application, open a terminal, navigate to your project directory, and use the following Maven command:

mvnw spring-boot:run

This will start the Spring Boot application on http://localhost:8080.

**11. Test with Postman**

**Add an Employee (POST Request):**

1. Open **Postman**.
2. Set the **HTTP method** to POST.
3. Enter the **URL**: http://localhost:8080/api/employees.
4. Go to the **Body** tab and select **raw** and **JSON**.
5. Enter the following JSON data in the body:

{  
 "id": "1",  
 "name": "John Doe",  
 "position": "Software Engineer",  
 "department": "IT"  
}

1. Click **Send** to add the employee.

**List Employees (GET Request):**

1. Set the **HTTP method** to GET.
2. Enter the **URL**: http://localhost:8080/api/employees.
3. Click **Send** to view the list of employees. The response should be:

[  
 {  
 "id": "1",  
 "name": "John Doe",  
 "position": "Software Engineer",  
 "department": "IT"  
 }  
]

**12. Conclusion**

You now have a working Spring Boot application for managing employee records. The application supports:

* Adding new employees via a POST request.
* Listing all employees via a GET request.