✅Case Study Title: Employee Info API using Spring Boot AutoConfiguration

🎯 **Objective:**

To build a simple Spring Boot application that exposes an API endpoint to retrieve basic employee information using **Spring Boot AutoConfiguration**. The endpoint will be tested via a browser and

💡 **Scenario:**

You are a developer working in the HR software team. Your task is to expose employee information (like name, ID, and department) through a simple HTTP GET API without manually configuring any server, servlet, or web.xml file.

Expected JSON output:

```
{
  "id": 101,
  "name": "John Doe",   "department": "Engineering"
}
```

SOLUTION :

Step 1: Create a Spring Boot Project

1. **Open Spring Tool Suite (STS)**
2. **Go to File → New → Spring Starter Project**
3. **Fill in the project metadata:**
   - **Name: employee-api**
   - **Type: Maven**
   - **Packaging: Jar**
   - **Java Version: 17 or 21 (choose based on your JDK)**
   - **Group: com.company**
   - **Artifact: employee-api**
   - **Click Next**
4. **Select Dependencies:**
   Spring Web
5. **Click Finish**

Step 2: Create the Model Class

File: Employee.java

```
package com.company.employeeapi.model;

public class Employee {
    private int id;
    private String name;
    private String department;

    // Constructors
    public Employee() {}

    public Employee(int id, String name, String department) {
        this.id = id;
        this.name = name;
        this.department = department;
```

```
    }

    // Getters
    public int getId() { return id; }
    public String getName() { return name; }
    public String getDepartment() { return department; }

    // Setters
    public void setId(int id) { this.id = id; }
    public void setName(String name) { this.name = name; }
    public void setDepartment(String department) { this.department = department; }
}
```

Step 3: Create the REST Controller
File: EmployeeController.java

```
package com.company.employeeapi.controller;

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

import com.company.employeeapi.model.Employee;

@RestController
public class EmployeeController {

    @GetMapping("/employee")
    public Employee getEmployee() {
        return new Employee(101, "Alice Johnson", "HR");
    }
}
```

Step 4: Run the Application
File: EmployeeApiApplication.java

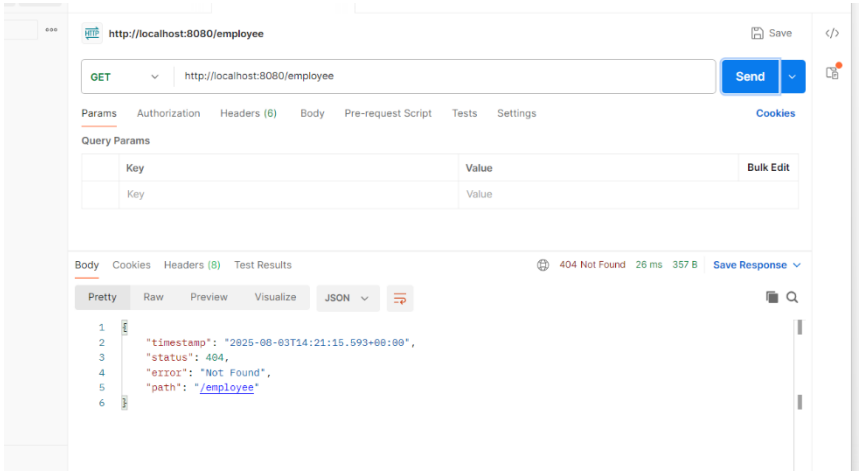
```
package com.company.employeeapi;

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

@SpringBootApplication
public class EmployeeApiApplication {

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

Run It:

- **Right-click EmployeeApiApplication.java**
- **Choose Run As → Spring Boot App**

Step 5: Test in Postman
http://localhost:8080/employee

```
{
 "id": 101,
 "name": "Alice Johnson",
 "department": "HR"
}
```



✅ 2. Spring Boot – Actuators

🎯 Case Study: Monitoring an Inventory System

Problem Statement:
You deploy an Inventory Management app and want to **monitor** its health, memory usage, bean loading, and environment settings without building these endpoints manually.

📝 Scenario:
You add the spring-boot-starter-actuator dependency, and enable the / actuator endpoint in application.properties.

SOLUTION :

Step 1: Create a New Spring Boot Project

- **Open STS**
- **Go to File → New → Spring Starter Project**
- **Fill in project details:**
  - **Name: inventory-monitor**
  - **Group: com.inventory**
  - **Artifact: inventory-monitor**
  - **Type: Maven**
  - **Packaging: Jar**
  - **Java Version: 17 (or 21 if your JDK supports it)**
  - **Click Next**
- **Select dependencies:**
  - **Spring Web**
  - **Spring Boot Actuator**

- **Click Finish**

Step 2:
File: InventoryController.java

```
package com.inventory.inventorymonitor;

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

@RestController
public class InventoryController {

   @GetMapping("/inventory")
   public String checkInventory() {
      return "Inventory is available!";
   }
}
```

Step 3: Configure application.properties
Open src/main/resources/application.properties and add:
properties

```
# Enable full actuator endpoint access
management.endpoints.web.exposure.include=*

# Optional: change server port if needed
# server.port=8081
```

Step 4: Run the Application

- **Right-click InventoryMonitorApplication.java**
- **Select Run As → Spring Boot App**

Step 5: Access Actuator Endpoints
Open Postman

- **http://localhost:8080/inventory → Your custom endpoint**
- **http://localhost:8080/actuator → Shows list of all actuator endpoints**
- **http://localhost:8080/actuator/health → Shows application health**
- **http://localhost:8080/actuator/beans → Lists Spring Beans**
- **http://localhost:8080/actuator/metrics → Shows JVM & HTTP metrics**
- **http://localhost:8080/actuator/env → Displays current environment properties**