Case Study Title: Employee Info API using Spring Boot AutoConfiguration

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 Postman using only @GetMapping.

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);

  }

 }
```

EmployeeController.java:

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

 import com.company.employeeapi.model.Employee;

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

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

@RestController

public class EmployeeController {

    @GetMapping("/employee")

    public Employee getEmployeeInfo() {

         return new Employee(101, "John Doe", "Engineering");

  }

 }
```

Employee.java:

```
package com.company.employeeapi.model;
 public class Employee {
     private int id;
     private String name;
     private String department;
     public Employee() {
    }
     public Employee(int id, String name, String department) {
         this.id = id;
         this.name = name;
        this.department = department;
     }
       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 getDepartment() { return department; }
      public void setDepartment(String department) { this.department = department; }
     }
```

Application.properties:

spring.application.name=employee-api

server.port=8080

Case study2: Spring Boot – Actuators

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.

InventoryMonitoringApplication.java:

```
package com.company.inventorymonitoring;

import org.springframework.boot.SpringApplication;

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

public class InventoryMonitoringApplication {

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

    }

}
```

application.properties:

```
spring.application.name=inventory-monitoring
management.endpoints.web.exposure.include=*

server.port=8082
```

InventoryMonitoringApplicationTests.java:

```
package com.company.inventory_monitoring;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
```

```
class InventoryMonitoringApplicationTests {

        @Test

        void contextLoads() {

    }

 }
```

Pom.xml: