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 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 org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Map;
@RestController
public class EmployeeController {
    @GetMapping("/employee")
    public Map<String, Object> getEmployee() {
        return Map.of(
                "id", 101,
                "name", "John Doe",
                "department", "Engineering"
```

```
        );
    }
}
```

http://localhost:8080/employee

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

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.

//OrderService

```
package com.example.service;

import org.springframework.stereotype.Service;

@Service

public class OrderService {

    public void addToCart(String product) {

        System.out.println("Adding product to cart: " + product);

    }

    public void placeOrder(String orderId) {

        if(orderId.equals("INVALID_ID")) {

            throw new RuntimeException("OrderNotFoundException");
```

```
        }
        System.out.println("Order placed successfully: " + orderId);
    }
    public void cancelOrder(String orderId) {
        System.out.println("Order cancelled: " + orderId);
    }
}
```

//OrderLoggingAspect

```
package com.example.aspect;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;
@Aspect
@Component
public class OrderLoggingAspect {
    @Before("execution(* com.example.service.OrderService.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("Starting method: " + joinPoint.getSignature().getName() +
            " with arguments: " + java.util.Arrays.toString(joinPoint.getArgs()));
    }
    @AfterReturning(pointcut = "execution(* com.example.service.OrderService.*(..))", returning =
"result")
    public void logAfterReturning(JoinPoint joinPoint, Object result) {
        System.out.println("Method " + joinPoint.getSignature().getName() + " executed successfully.");
    }
    @AfterThrowing(pointcut = "execution(* com.example.service.OrderService.*(..))", throwing =
"ex")
    public void logAfterThrowing(JoinPoint joinPoint, Throwable ex) {
        System.out.println("Exception in method: " + joinPoint.getSignature().getName() +
            " with message: " + ex.getMessage());
    }
```

```
    @After("execution(* com.example.service.OrderService.*(..))")
    public void logAfter(JoinPoint joinPoint) {
        System.out.println("Method " + joinPoint.getSignature().getName() + " execution finished.");
    }
}
```

//application.properties

```
# Server configuration
server.port=8080
# Enable all actuator endpoints
management.endpoints.web.exposure.include=*
# Customize health status
management.endpoint.health.show-details=always

# Optional: base path for all actuator endpoints
management.endpoints.web.base-path=/actuator
# Logging level (optional, useful for AOP logging visibility)
logging.level.org.springframework.aop=DEBUG
logging.level.com.yourpackage=DEBUG
```

//pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>order-inventory-system</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Order Inventory System</name>
  <description>Spring Boot AOP and Actuator Case Study</description>
  <packaging>jar</packaging>
```

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.1.5</version> <!-- Use latest compatible version -->
  <relativePath/> <!-- lookup parent from repository -->
</parent>
<properties>
  <java.version>17</java.version> <!-- or 11 depending on your setup -->
</properties>
<dependencies>
  <!-- Core Spring Boot Starter -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <!-- Spring Boot AOP -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
  <!-- Spring Boot Actuator -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <!-- Optional: Spring Web for REST endpoints -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

```
    <!-- Test dependencies -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- Spring Boot Maven Plugin -->
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```