**WEEK 4&5**

**EXERCISE 11**

**Online Bookstore - Integrating Spring Boot Actuator**

**Business Scenario**

The task is to monitor and manage the bookstore's RESTful services using Spring Boot Actuator. Actuator provides a set of built-in endpoints to monitor and manage the application, along with the ability to expose custom metrics for more detailed insights.

**Instructions**

**1. Add Actuator Dependency:**

**Task:** Include the Spring Boot Actuator dependency in your project.

**Implementation:**

* Add the Actuator Dependency to Your pom.xml or build.gradle:

**For Maven:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

**For Gradle:**

implementation 'org.springframework.boot:spring-boot-starter-actuator'

**Refresh the Project:** After adding the dependency, refresh the project to download and integrate the Actuator library.

**2. Expose Actuator Endpoints:**

**Task:** Enable and customize Actuator endpoints.

**Implementation:**

* Configure Endpoints in application.properties or application.yml:

**For application.properties:**

management.endpoints.web.exposure.include=health,info,metrics

management.endpoint.health.show-details=always

management.endpoints.web.base-path=/actuator

**For application.yml:**

management:

endpoints:

web:

exposure:

include: health,info,metrics

base-path: /actuator

endpoint:

health:

show-details: always

* **Explanation:**
  + management.endpoints.web.exposure.include specifies which endpoints to expose.
  + management.endpoint.health.show-details controls whether detailed health information is shown.
  + management.endpoints.web.base-path sets the base path for Actuator endpoints.
* **Access Endpoints:**
  + **Health:** http://localhost:8080/actuator/health
  + **Info:** http://localhost:8080/actuator/info
  + **Metrics:** http://localhost:8080/actuator/metrics

**3. Custom Metrics:**

**Task:** Expose custom metrics for monitoring your application.

**Implementation:**

* Create a Custom Metric Configuration:

**CustomMetricsConfig.java:**

import io.micrometer.core.instrument.MeterRegistry;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class CustomMetricsConfig {

@Bean

public CustomMetrics customMetrics(MeterRegistry meterRegistry) {

return new CustomMetrics(meterRegistry);

}

public static class CustomMetrics {

private final MeterRegistry meterRegistry;

public CustomMetrics(MeterRegistry meterRegistry) {

this.meterRegistry = meterRegistry;

// Register custom metrics

meterRegistry.counter("custom\_metric\_counter", "type", "example");

}

}

}

**Access Custom Metrics:**

Once the custom metrics are registered, they will be available in the /actuator/metrics endpoint. You can view them with the URL:

http://localhost:8080/actuator/metrics/custom\_metric\_counter

**Explanation:**

* **MeterRegistry:** Used to register custom metrics with the application.
* **Custom Metrics Class:** Defines and registers metrics, such as counters, gauges, and timers.

**Testing:**

* **Verify Actuator Endpoints:** Access the exposed endpoints via a web browser or API client to ensure they are working.
* **Monitor Custom Metrics:** Check the metrics endpoint to verify that your custom metrics are being reported correctly.

**Conclusion:**

Integrating Spring Boot Actuator provides valuable insights into the health, performance, and behaviour of the application. Custom metrics allow for tailored monitoring to better understand and manage the application's performance.