**Exercise 11: Online Bookstore - Integrating Spring Boot Actuator**

Business Scenario:

Monitor and manage your bookstore's RESTful services using Spring Boot Actuator.

**Adding Spring Boot Actuator Dependency:**

To integrate Spring Boot Actuator, you need to add the Actuator dependency in your pom.xml (for Maven projects). This dependency enables various monitoring and management features in your Spring Boot application.

**Exposing Actuator Endpoints:**

By default, Spring Boot Actuator exposes a variety of endpoints that provide information about your application (e.g., /health, /metrics). You can customize which endpoints are enabled and whether they require authentication.

**Custom Metrics:**

Spring Boot Actuator allows to define custom metrics to monitor specific aspects of your application. For example, you may want to track the number of book purchases or customer registrations.

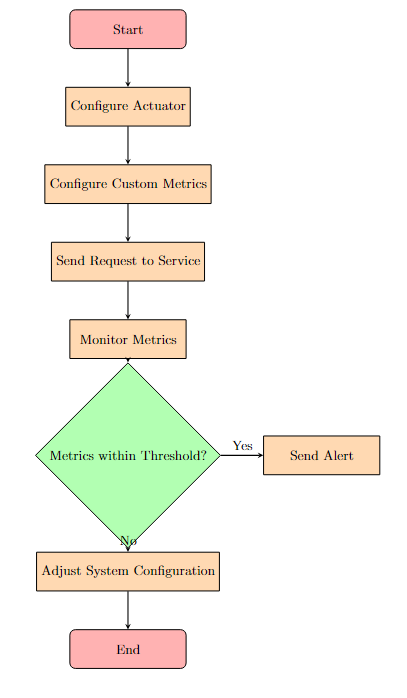
**Accessing Actuator Endpoints:**

We can access the Actuator endpoints via HTTP by navigating to specific URLs. These endpoints provide valuable insights into your application's performance, health, and other operational metrics.

**Key Points:**

* /actuator/health: Displays the health status of your application.
* /actuator/metrics: Shows various metrics, including custom metrics defined by you.
* /actuator/env: Provides details about the application's environment variables.

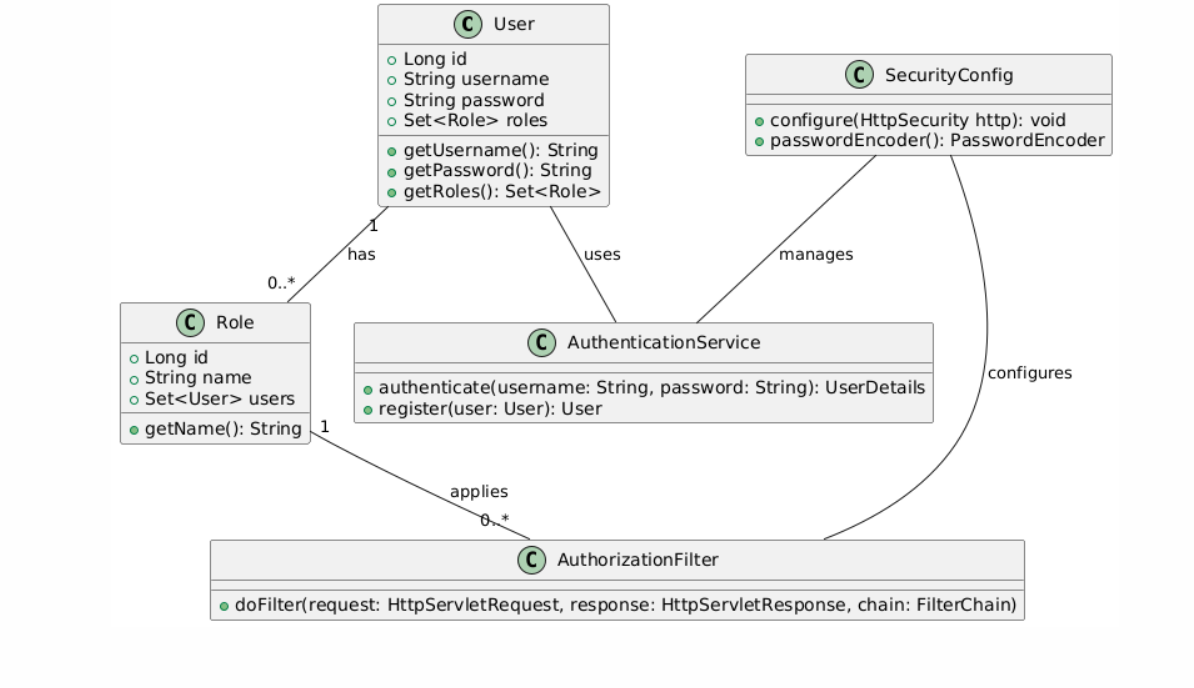
**FLOWCHART:**



**Explanation:**

1. **Start Node:**
   * The flowchart begins with a "Start" node, indicating the beginning of the process.
2. **Configure Actuator:**
   * This process involves configuring Spring Boot Actuator in your application by including the necessary dependencies and enabling the monitoring endpoints.
3. **Configure Custom Metrics:**
   * This step involves defining custom metrics using MeterRegistry, allowing you to track specific aspects of your application, such as custom counters or timers.
4. **Send Request to Service:**
   * In this step, a request is sent to the RESTful service, which can be a GET, POST, PUT, or DELETE request.
5. **Monitor Metrics:**
   * After the request is sent, Spring Boot Actuator collects metrics, including custom ones, that help in monitoring the performance and health of the services.
6. **Decision Node - Metrics within Threshold?**
   * At this decision point, the system checks whether the collected metrics are within the defined threshold (e.g., response time, error rate).
7. **Yes - Send Alert:**
   * If the metrics are not within the acceptable threshold, an alert is triggered. This can involve sending notifications through email, SMS, or integrating with monitoring tools like Prometheus and Grafana.
8. **No - Adjust System Configuration:**
   * If the metrics are within the acceptable threshold, the system might need adjustments or configuration changes to ensure optimal performance.
9. **End Node:**
   * Finally, the flowchart ends, indicating that the process is complete.

**CLASS DIAGRAM :**



**Explanation:**

1. **User Class:**
   * This class represents the user entity, which has attributes like id, username, password, and a collection of roles (roles). Methods like getUsername(), getPassword(), and getRoles() provide access to these attributes.
2. **Role Class:**
   * The Role class represents a role or authority granted to users. It has attributes like id and name. The relationship between User and Role is many-to-many, meaning a user can have multiple roles, and a role can be assigned to multiple users.
3. **AuthenticationService Class:**
   * This service handles authentication logic, including methods for authenticating users (authenticate()) and registering new users (register()).
4. **AuthorizationFilter Class:**
   * This class represents a security filter that intercepts incoming HTTP requests and applies authorization logic. The method doFilter() processes the request and response based on security rules.
5. **SecurityConfig Class:**
   * The SecurityConfig class configures Spring Security, including HTTP security settings and password encoding. It uses the AuthorizationFilter to apply security rules and the AuthenticationService to handle authentication.
6. **Relationships:**
   * The User class has a one-to-many relationship with Role, indicating that each user can have multiple roles.
   * The AuthenticationService is used by the User class for authentication purposes.
   * The SecurityConfig class configures and manages both AuthorizationFilter and AuthenticationService.