**Coding Exercise:**

**Aim**

1. Create a REST endpoint whose invocation counter is updated atomically even if invoked by multithreaded clients.
2. Make application production ready by adding production monitoring. (using Spring Actuator)
3. API documentation (using Swagger)

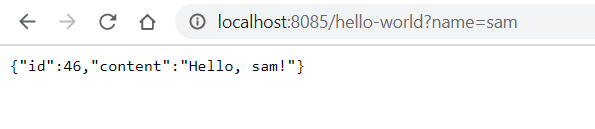
**Compilation and execution Instructions**

1. Run the application by using ./mvnw spring-boot:run.
2. Alternatively, build the fat JAR file with ./mvnw clean package and then run the JAR file, as follows:

java -jar target/demo-0.1.0.jar

**Spring REST Endpoint**

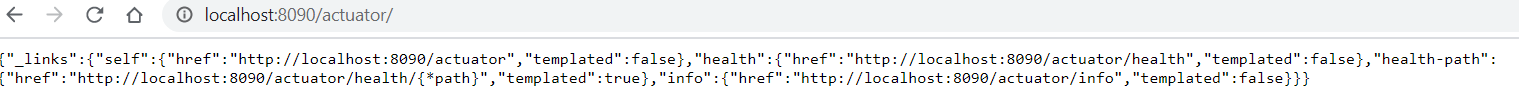
<http://localhost:8085/hello-world?name=sam>



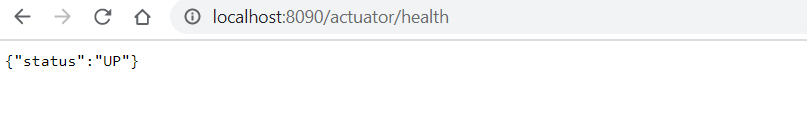
**Spring Application Monitoring (using spring actuator)**

**Endpoint:** <http://localhost:8090/actuator/>

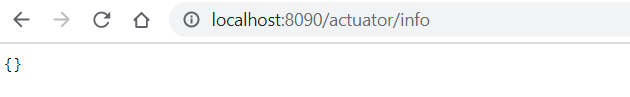
Output screenshot:



**Endpoint:** <http://localhost:8090/actuator/health>

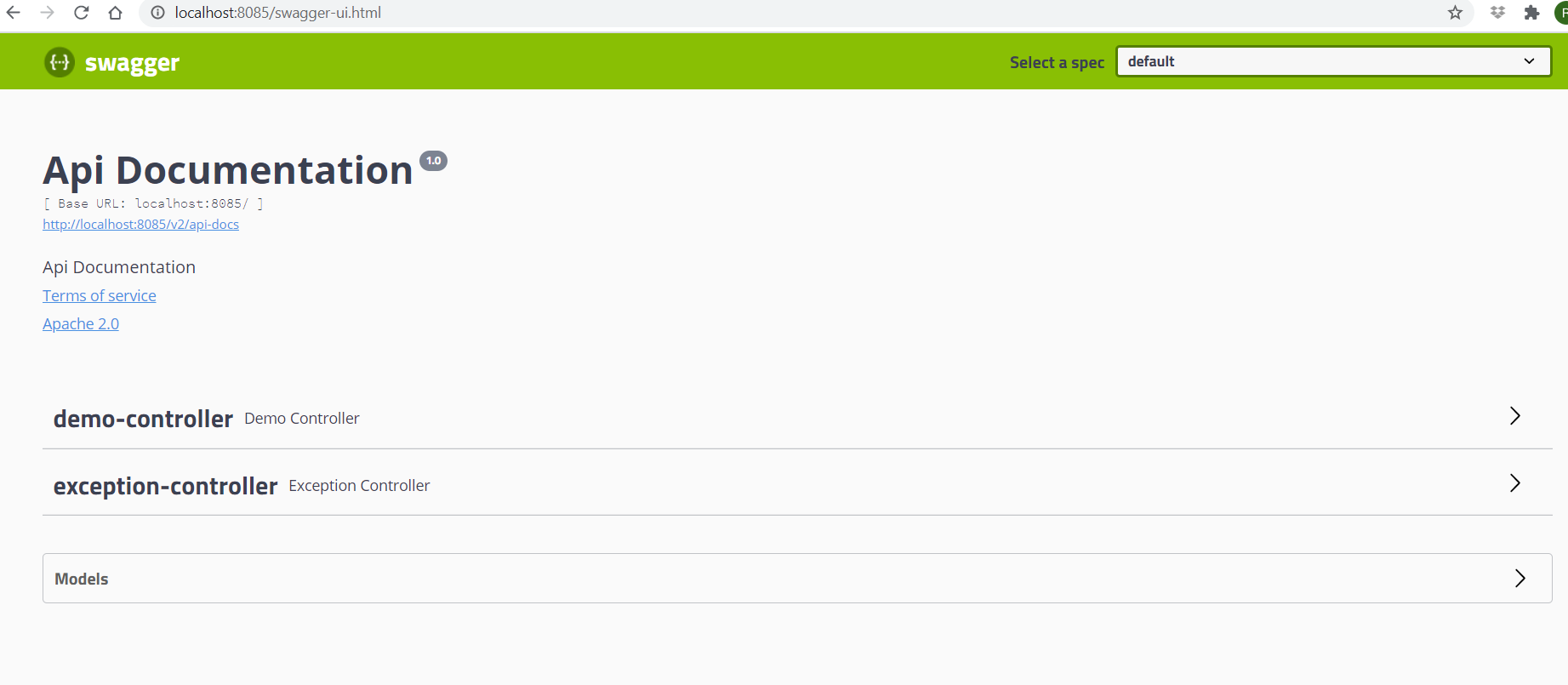


**Endpoint:** <http://localhost:8090/actuator/info>



**API documentation (using Swagger)**

<http://localhost:8085/swagger-ui.html>



**Troubleshooting** i**ssues found during coding exercise:**

1. Port 8080 used by another container and container not starting up.

**Resolution:** Modified “application.properties” as per below:

**server.port**: **8085  
management.server.port**: **8090  
management.server.address**: **127.0.0.1**

**Further enhancements:**

1. As part of CLEAN code, SOLID, DRY design principles philosophy, I have spent couple of hours but happy to extend it based on your advice.
2. Use libraries like below to test atomicity, concurrency and race condition issues:
   * concurrent-junit
   * vmlens
   * Thread Weaver
   * Tempus-fugit
3. Add tests using performance testing like JMeter to test API performance and throttling.
4. Improve exception handling for API.
   * Question: Should client handle HTTP error code like 404, 501 or API return status code?
5. Add API scalability, availability, reliability (using spring projects Spring Cloud Gateway etc) to throttle if API is overload.
6. Add API throttling, circuit breaker (using spring projects Spring Cloud Gateway, Hystrix, Ribbon etc) to throttle if API is overload.
7. Add API security (using Spring Security project).
8. Add deployment to cloud(example: using azure-webapp-maven-plugin**).**
9. Add API service registration and discovery ( example: Eureka service registry).