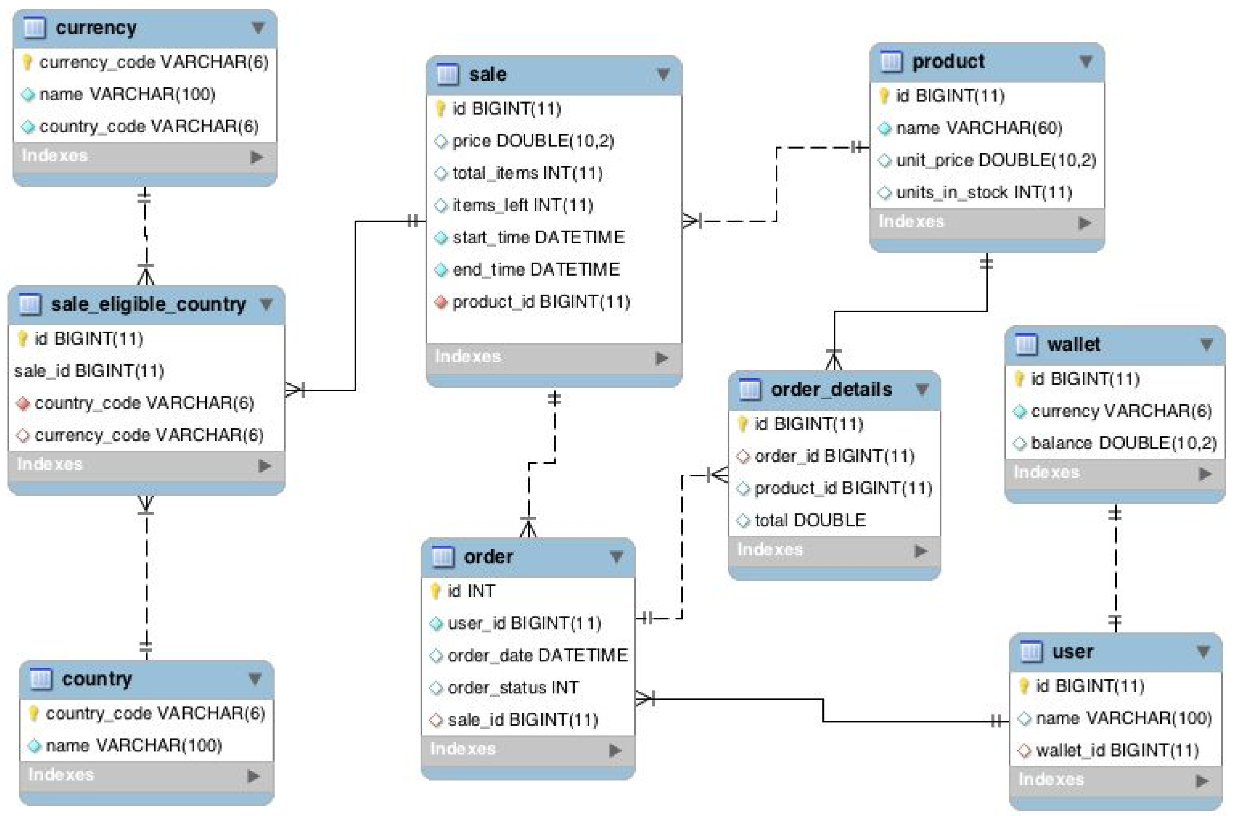
Shopping Societies E-Commerce Platform

Introduction

This document includes all the instructions required for running this project.
Also I have mentioned some of the assumptions which I made while developing APIs.

Entity Relationship Diagram (ERD) of Application



Assumptions Related to Api Endpoints

1.Endpoint: GET /sales/current?country=SG

Assumption: Include the **sale_id** in the response

Reason: Since the flash sale can be eligible for many countries , including **sale_id** when displaying flash sales will help to reduce the complexity of managing **purchase** API . Otherwise we need to pass the **"country"** along with **"user_id"** in /products/{id}/purchase

2. Endpoint: POST /products/{id}/purchase
Assumption: id = sales_id
Reason: As mentioned above

3.Company is also a user and its user_id is = 1

Application Development

Application has been developed using following components.

- Java 8 (version "1.8.0_162")
- Spring Boot framework
- Embedded web container
- MySQL as the database
- Flyway as database migration tool
- Maven build tool
- Spring Data JPA for handling database functions
- Junit and mockito for unit and integration testing
- Enabled docker if you need to run it in containerized environment
- Gatling for load testing
- Swagger UI for viewing API endpoint details
- Cobertura for code coverage

System Ports

By default system is running in following ports. Please change application and docker property files if there is any conflict with your environment

Web App: **8888**
MySQL: **3306**

System Requirements

Following libraries should be installed on your machine before running the application.

- Java 8+
- Maven
- MySqL (mysql@5.7 or latest)
- Docker (only if you if you need to run it in containerized environment)

Running the Application

Application can be run in different profiles such as test , dev, prod and docker. By default it will be running in prod profile. Therefore you may need to change the application property files based on selected profile.

Running in prod profile

Step 1

Create a MySQL database with the name " **prd_shopping_societies**" and grant all privileges for the app user.

Change the following properties in **/src/main/resources/application-prod.properties** accordingly.

```
## Spring DATA SOURCE Configurations
spring.datasource.url = jdbc:mysql://localhost:3306/prd_shopping_societies?useSSL=false
spring.datasource.username = app_user
spring.datasource.password = test123
```

Step 2

In Linux/Mac/Windows terminal run the following commands.
Alternatively you can run the project in any Java IDE.

Firstly, in the terminal , locate to the root folder of the application. Eg: **shoppingsocieties**

```
cd shoppingsocieties

mvn clean install
```

If you want to skip unit and integration testing

```
mvn clean install -DskipTests
```

If the build process is successful then you would see “BUILD SUCCESS” message. After that run the following command to start the application.

```
java -jar ./target/shoppingsocieties-0.0.1-SNAPSHOT.jar
```

If the the application started without any exceptions you would be able to access it from
http://localhost:8888/shoppingsocieties

Running in dev or test profile

Firstly you need to change the profile as **dev** in **/src/main/resources/application.properties**

Eg: dev profile

```
# test,dev , prod, docker
spring.profiles.active=dev
```

Change the Spring DATA SOURCE configurations in **application-dev.properties**

Then follow the same instructions mentioned in above **Step 2**

Running in test profile

Firstly you need to change the profile as **test** in **/src/main/resources/application.properties**

Eg: dev profile

```
# test,dev , prod, docker
spring.profiles.active=test
```

Change the Spring DATA SOURCE configurations in **application-test.properties**

Then follow the same instructions mentioned in above **Step 2**

Running in docker profile

Firstly you need to change the profile as **docker** in
/src/main/resources/application.properties
Check the docker-compose and Docker files to avoid any port conflicts with your loca environment.
From the application's root directory run the following commands.

```
mvn clean install -DskipTests

docker-compose up
```

If everything went successfully you would be able to access application from

http://localhost:8888/shoppingsocieties

Note: If you changed the application code or docker-compose.yml file you may need to rebuild.

```
mvn clean install -DskipTests
docker-compose rebuild
docker-compose up
```

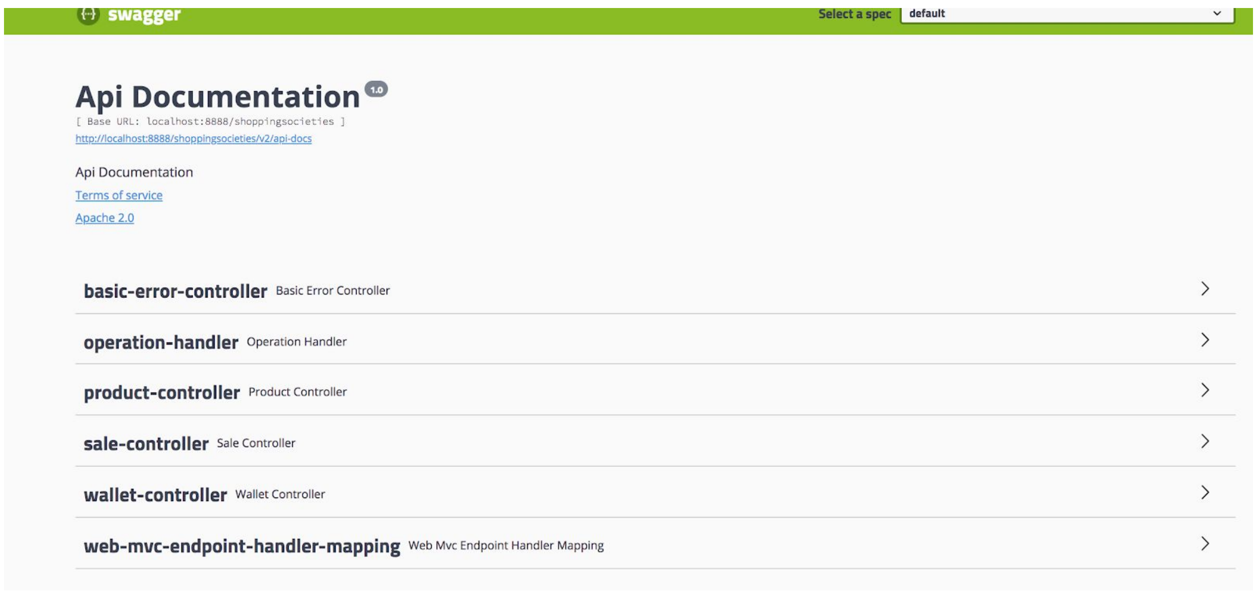
Testing the Application

Testing REST Endpoints Using Swagger UI

Swagger has been enabled for this application and the endpoints can be tested by using it. All the endpoints can be viewed using following swagger-ui URL

http://localhost:8888/shoppingsocieties/swagger-ui.html#/

Alternatively you can select any other rest client for testing endpoints.



Api Documentation 1.0

[Base URL: localhost:8888/shoppingsocieties]

http://localhost:8888/shoppingsocieties/v2/api-docs

Api Documentation

Terms of service

Apache 2.0

basic-error-controller Basic Error Controller

operation-handler Operation Handler

product-controller Product Controller

sale-controller Sale Controller

wallet-controller Wallet Controller

web-mvc-endpoint-handler-mapping Web Mvc Endpoint Handler Mapping

Running Unit and Integration Test

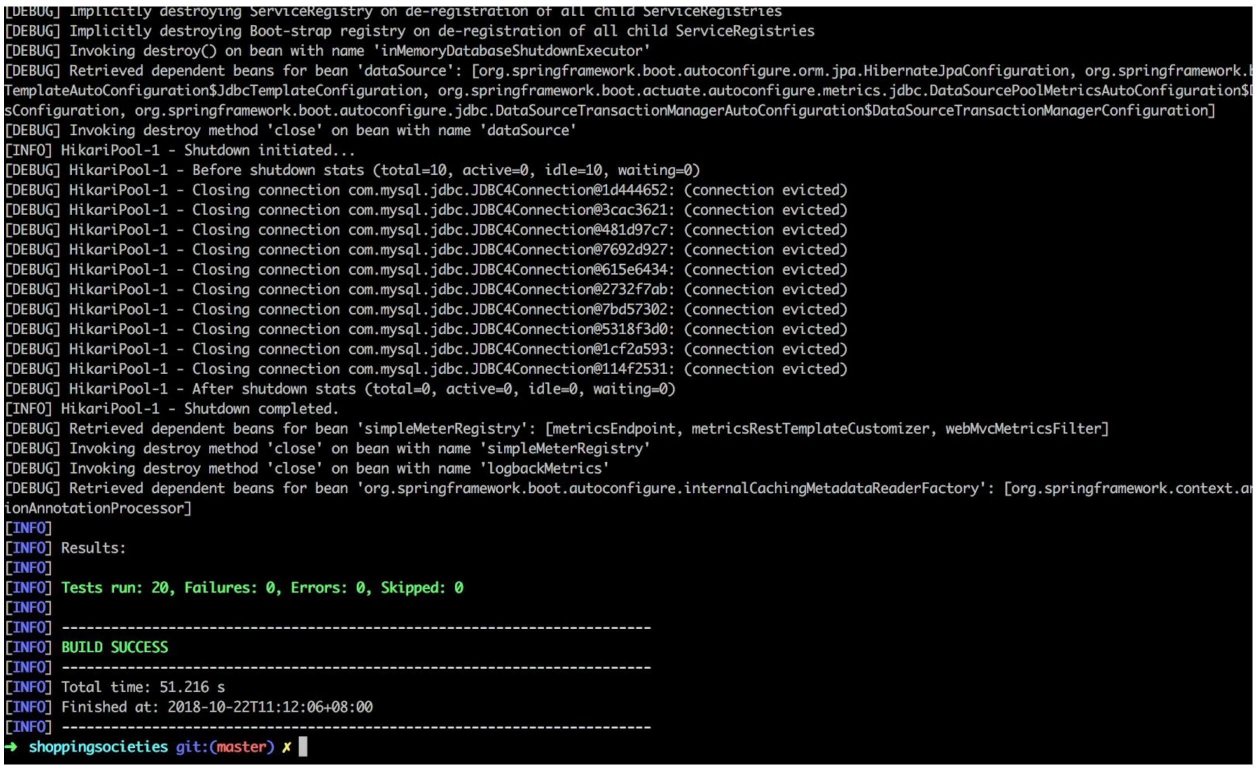
Unit and Integration test cases have been written using Junit and Mockito.
At the moment all the test cases are running together.

Firstly you need to change the profile as **test** in
/src/main/resources/application.properties

Since integration test has required a database , create a database and change the Spring DATA SOURCE configurations in **application-test.properties**

In the terminal, go to the root folder of the application and run the following command.

```
mvn  clean test
```



```
[DEBUG] Implicitly destroying ServiceRegistry on de-registration of all child ServiceRegistries
[DEBUG] Implicitly destroying Boot-strap registry on de-registration of all child ServiceRegistries
[DEBUG] Invoking destroy() on bean with name 'inMemoryDatabaseShutdownExecutor'
[DEBUG] Retrieved dependent beans for bean 'dataSource': [org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaConfiguration, org.springframework.
TemplateAutoConfiguration$JdbcTemplateConfiguration, org.springframework.boot.actuate.autoconfigure.metrics.jdbc.DataSourcePoolMetricsAutoConfiguration$
sConfiguration, org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration$DataSourceTransactionManagerConfiguration]
[DEBUG] Invoking destroy method 'close' on bean with name 'dataSource'
[INFO] HikariPool-1 - Shutdown initiated...
[DEBUG] HikariPool-1 - Before shutdown stats (total=10, active=0, idle=10, waiting=0)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@1d444652: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@3cac3621: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@481d97c7: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@7692d927: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@615e6434: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@2732f7ab: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@7bd57302: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@5318f3d0: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@1cf2a593: (connection evicted)
[DEBUG] HikariPool-1 - Closing connection com.mysql.jdbc.JDBC4Connection@114f2531: (connection evicted)
[DEBUG] HikariPool-1 - After shutdown stats (total=0, active=0, idle=0, waiting=0)
[INFO] HikariPool-1 - Shutdown completed.
[DEBUG] Retrieved dependent beans for bean 'simpleMeterRegistry': [metricsEndpoint, metricsRestTemplateCustomizer, webMvcMetricsFilter]
[DEBUG] Invoking destroy method 'close' on bean with name 'simpleMeterRegistry'
[DEBUG] Invoking destroy method 'close' on bean with name 'logbackMetrics'
[DEBUG] Retrieved dependent beans for bean 'org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory': [org.springframework.context.a
ionAnnotationProcessor]
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 20, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 51.216 s
[INFO] Finished at: 2018-10-22T11:12:06+08:00
[INFO] ------------------------------------------------------------------------
➜ shoppingsocieties git:(master) ✗
```

Running Load Test

In order to run the load testing , I have created a separate Spring Boot application and integrated Gatling performance testing tool.

However time was not enough to implement real scenario based load test cases. Following are the instructions for running some basic test cases.

Simulation of Current Flash Sales

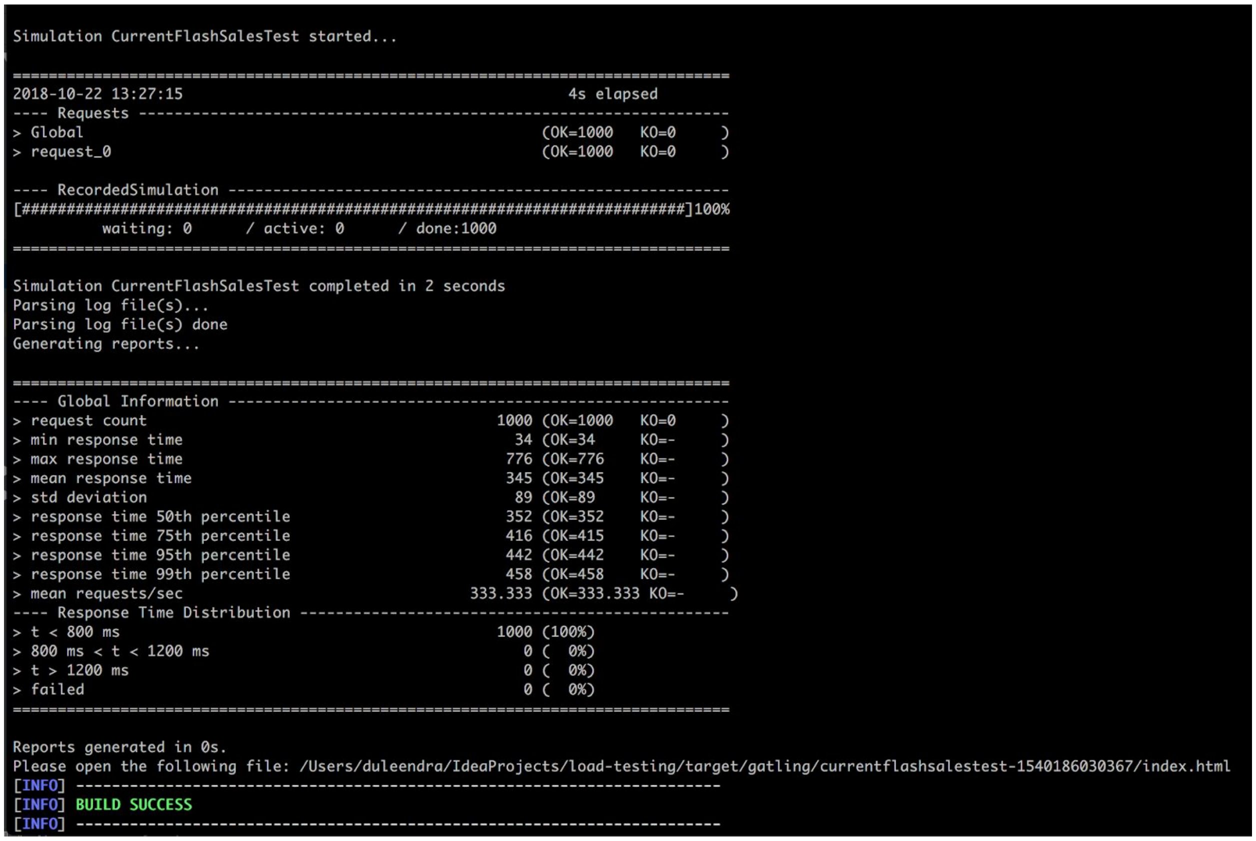
Load test endpoint: http://localhost:8888/shoppingsocieties/sales/current?country=SG

Steps

1. Go to the **load-testing** spring boot application folder.
2. Make sure **shoppingsocieties** web application is running before executing commands.
3. From the root folder run the following commands in the terminal.

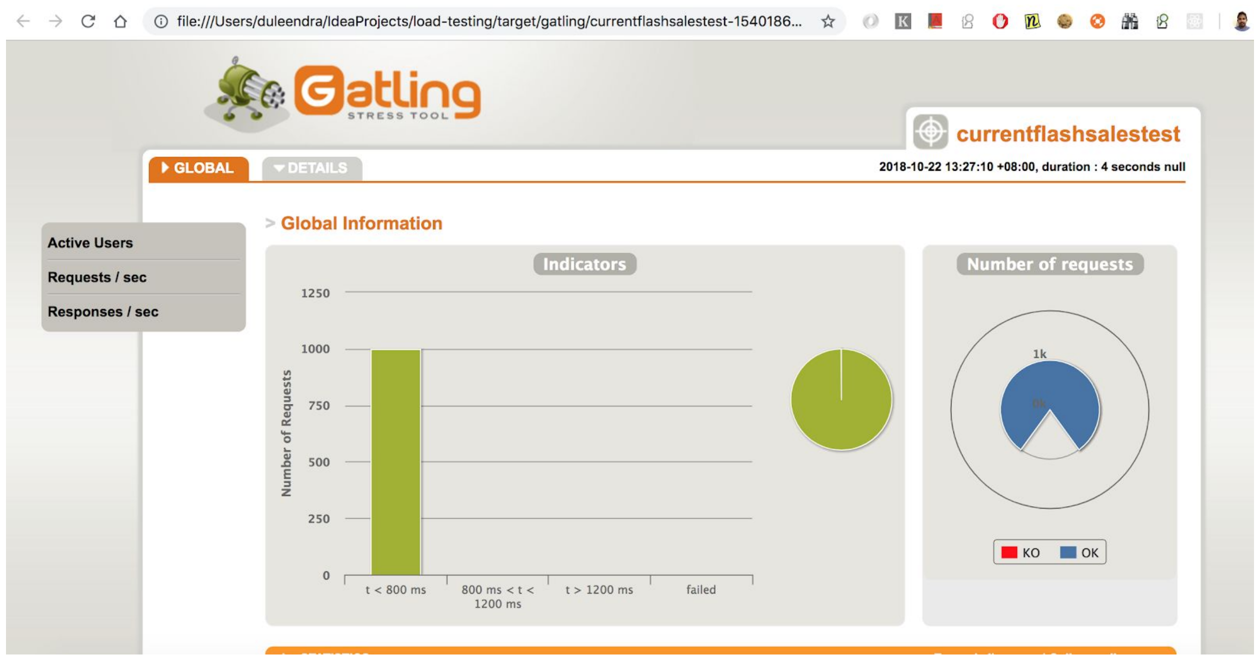
```
mvn clean install
mvn gatling:test
```

If everything went successfully you would see the following output.
Also you can see the full report from target/gatling/currentflashsalestest*/index.html

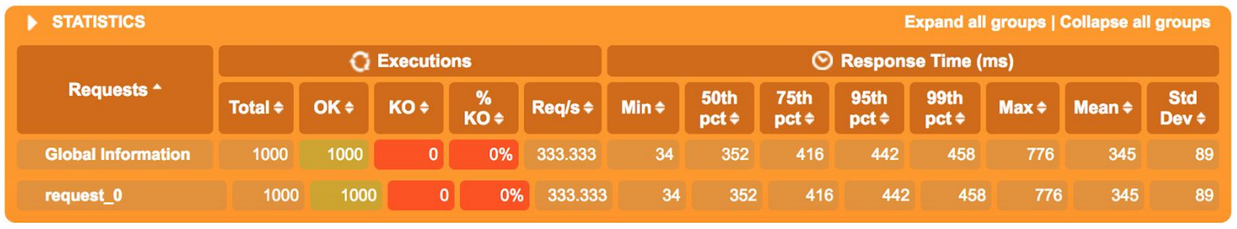


```
Simulation CurrentFlashSalesTest started...

================================================================================
2018-10-22 13:27:15                                           4s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=1000   KO=0     )
> request_0                                                (OK=1000   KO=0     )

---- RecordedSimulation --------------------------------------------------------
[##########################################################################]100%
          waiting: 0      / active: 0      / done:1000
================================================================================

Simulation CurrentFlashSalesTest completed in 2 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information --------------------------------------------------------
> request count                                       1000 (OK=1000   KO=0     )
> min response time                                     34 (OK=34     KO=-     )
> max response time                                    776 (OK=776    KO=-     )
> mean response time                                   345 (OK=345    KO=-     )
> std deviation                                         89 (OK=89     KO=-     )
> response time 50th percentile                        352 (OK=352    KO=-     )
> response time 75th percentile                        416 (OK=415    KO=-     )
> response time 95th percentile                        442 (OK=442    KO=-     )
> response time 99th percentile                        458 (OK=458    KO=-     )
> mean requests/sec                                333.333 (OK=333.333 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                          1000 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================

Reports generated in 0s.
Please open the following file: /Users/duleendra/IdeaProjects/load-testing/target/gatling/currentflashsalestest-1540186030367/index.html
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

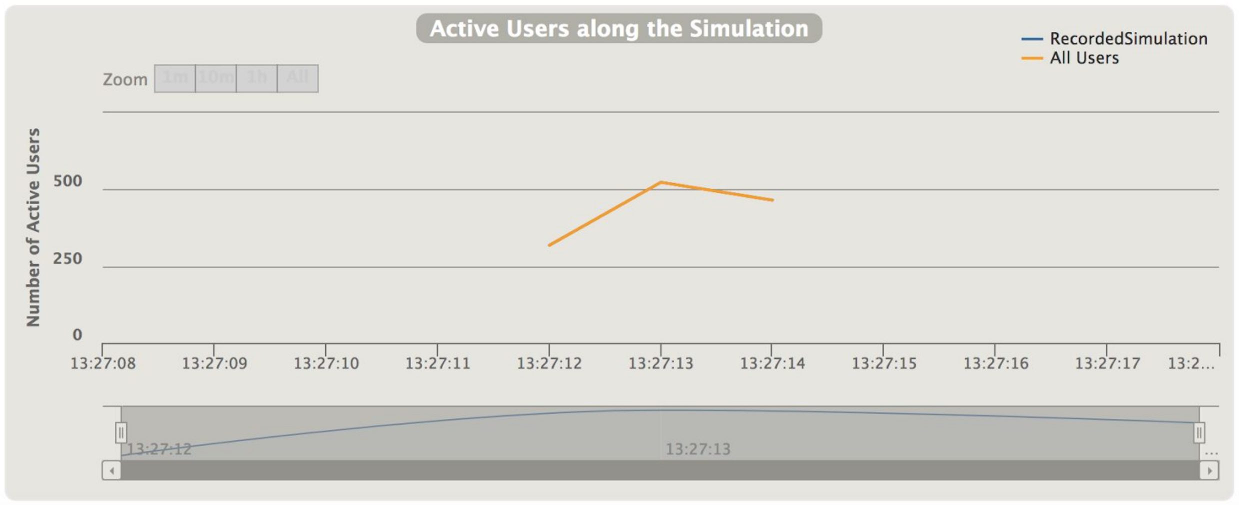
Sample outputs


file:///Users/duleendra/IdeaProjects/load-testing/target/gatling/currentflashsalestest-1540186...

2018-10-22 13:27:10 +08:00, duration : 4 seconds null

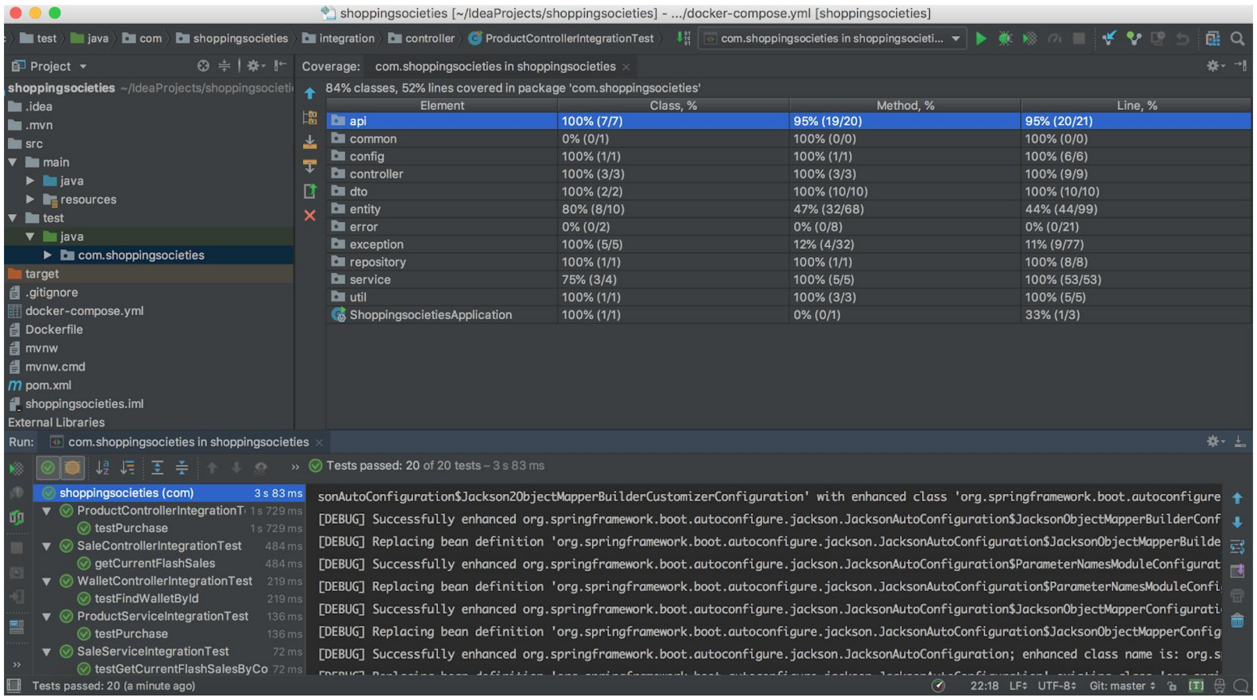

STATISTICS   Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	1000	1000	0	0%	333.333	34	352	416	442	458	776	345	89
request_0	1000	1000	0	0%	333.333	34	352	416	442	458	776	345	89



Code Coverage

100% Unit test coverage has been given for all the Service, and Rest Controller classes.



shoppingsocieties [~/IdeaProjects/shoppingsocieties] - .../docker-compose.yml [shoppingsocieties]

Coverage: com.shoppingsocieties in shoppingsocieties

84% classes, 52% lines covered in package 'com.shoppingsocieties'

Element	Class, %	Method, %	Line, %
api	100% (7/7)	95% (19/20)	95% (20/21)
common	0% (0/1)	100% (0/0)	100% (0/0)
config	100% (1/1)	100% (1/1)	100% (6/6)
controller	100% (3/3)	100% (3/3)	100% (9/9)
dto	100% (2/2)	100% (10/10)	100% (10/10)
entity	80% (8/10)	47% (32/68)	44% (44/99)
error	0% (0/2)	0% (0/8)	0% (0/21)
exception	100% (5/5)	12% (4/32)	11% (9/77)
repository	100% (1/1)	100% (1/1)	100% (8/8)
service	75% (3/4)	100% (5/5)	100% (53/53)
util	100% (1/1)	100% (3/3)	100% (5/5)
ShoppingsocietiesApplication	100% (1/1)	0% (0/1)	33% (1/3)

Run: com.shoppingsocieties in shoppingsocieties

Tests passed: 20 of 20 tests – 3 s 83 ms

```
shoppingsocieties (com)                       3 s 83 ms
  ProductControllerIntegrationT               1 s 729 ms
    testPurchase                              1 s 729 ms
  SaleControllerIntegrationTest               484 ms
    getCurrentFlashSales                      484 ms
  WalletControllerIntegrationTest             219 ms
    testFindWalletById                        219 ms
  ProductServiceIntegrationTest               136 ms
    testPurchase                              136 ms
  SaleServiceIntegrationTest                  72 ms
    testGetCurrentFlashSalesByCo              72 ms
```

```
sonAutoConfiguration$Jackson2ObjectMapperBuilderCustomizerConfiguration' with enhanced class 'org.springframework.boot.autoconfigure
[DEBUG] Successfully enhanced org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObjectMapperBuilderConf
[DEBUG] Replacing bean definition 'org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObjectMapperBuilde
[DEBUG] Successfully enhanced org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$ParameterNamesModuleConfigurat
[DEBUG] Replacing bean definition 'org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$ParameterNamesModuleConfi
[DEBUG] Successfully enhanced org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObjectMapperConfigurati
[DEBUG] Replacing bean definition 'org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObjectMapperConfig
[DEBUG] Successfully enhanced org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration; enhanced class name is: org.s
```

Tests passed: 20 (a minute ago)