Case Study: myretail RESTful service

Table of Contents

[Approach: Architecture 1](#_Toc142253734)

[API Specification 2](#_Toc142253735)

[Technology and Framework choices 3](#_Toc142253736)

[Security 3](#_Toc142253737)

[Prerequisite List 3](#_Toc142253738)

[Building, Testing and Executing the Code 4](#_Toc142253739)

[Manual Testing using Insomnia 5](#_Toc142253740)

[Performance testing 5](#_Toc142253741)

# Approach: Architecture

A diagram of a product

Description automatically generated

Ideally, we will have 3 distinct standalone APIs with their own ecosystem.

But for the case study we have one project with all 3 APIs [Controller/Service/Rep layers] for convenience of demoing.

A screenshot of a blue screen

Description automatically generated

# API Specification

Please refer to /api- spec/myretail\_spec-v1.yml, paste the spec into https://editor.swagger.io, to review.

Callouts:

1. Key and Auth Token security enforced – Price is sensitive information, for the demo both must be passed.
2. Prices could have multiple variations, eg: Current price, Retail price or Initial price. Hence the PUT is to update a specific price\_type, and returns the updated state of the entire Price Document related to the product id.

# Technology and Framework choices

|  |  |
| --- | --- |
| Technology | Comments/Reason for choosing |
| Kotlin JVM | Java Version 17 |
| Retrofit | Robust way to integrate with partner APIs, it abstracts out the complexity of building the Request using the correct URL Properties, adding the Key and Auth token to the headers. It keeps the service layer code cleaner and more maintainable. |
| Spock + Groovy | Unit and Integration Test |
| Jacoco | Code coverage – Refer in build.gradle |
| Springboot | Familiar Framework, and provide seamless integrations with Mongo, with Zero boiler plate |
| Docker | Docker containerization for Mongo DB for integration tests and running the application live for the demo |
| MongoDB + Mongock | NoSQL DB, Provides Consistency and Partition Tolerance. Mongock is a way to manage document schema versioning, indexes |
| JMeter | Performance testing the APIs |
| Swagger.io | API Documentation |
| Insomnia | Manual API Testing |

# Security

Basic simple custom security setup

Created a simple set of Key and Bearer Token Request Filters, just for this case study, so that the application itself is self-contained for demo purposes.

# Prerequisite List

|  |  |
| --- | --- |
| 1 | Your machine has Java 17 |
| 2 | Docker is installed and is running |
| 3 | Using docker you have been able to pull the latest version of Mongo |
| 4 | You have MongoDB Compass installed |
| 5 | You have JMeter locally installed |
| 6 | Insomnia is installed |

# Building, Testing and Executing the Code

|  |  |
| --- | --- |
| Instantiate mongo in docker | $ docker run -d -p 27017:27017 --name myretail-db -e MONGO\_INITDB\_ROOT\_USERNAME=admin -e MONGO\_INITDB\_ROOT\_PASSWORD=password mongo:latest |
| Code – clean and build | Unzip myretail.zip to a folder of your choice $ cd myretail  $./gradlew clean build  You should see the following.  **BUILD SUCCESSFUL** in 17s  28 actionable tasks: 28 execute |
| Jacoco Test Coverage Report | $cd /myretail/build/jacocoHtmlReport  Open index.html for coverage report |
| Unit Test Report | $cd /myretail/build/reports/tests/test  Open index.html for unit test report |
| Integration Test Report | $cd /myretail/build/reports/tests/ integrationTest  Open index.html for integration test report |
| Executing the Code | $ ./gradlew clean bootRun --args='--spring.profiles.active=prod --spring.data.mongodb.uri=mongodb://admin:password@localhost:27017/myretail-db?authSource=admin'  You should see the following.  . \_\_\_\_ \_ \_\_ \_ \_  /\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \  ( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  \\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /  =========|\_|==============|\_\_\_/=/\_/\_/\_/  :: Spring Boot :: (v2.7.13)  ...  INFO: Initializing Spring embedded WebApplicationContext  **<===========--> 90% EXECUTING [1m 9s]**  **> :bootRun** |
| Connecting to Mongo Compass | Open Mongo Compass using the following connection string: mongodb://admin:password@localhost:27017/myretail-db?authSource=admin |
| Validate Mongo Collections | In compass check the ‘prices’ collection, and check the ‘indexes’ tab to see the predefined index on ‘product id’, and in the ‘validations’ tab you can see the clear definition of the format a document should have, to be saved in this collection |

# Manual Testing using Insomnia

Open insomnia and import collection from path /myretail/insomnia/myretail\_collections.json

Run all the GET, POST, PUT requests to validate requirement of the case study.

# Performance testing

Goto folder /myretail/perf and make it your pwd. And start jmeter in your cli, and when the GUI comes up select ‘myretail\_perf\_tests.jmx’ and import the test suite.

The last run resulted in the following data

|  |  |  |
| --- | --- | --- |
| Action | Sample | Throughput |
| POST - Create Prices for Product | 2487 | 41.6/sec |
| PUT - Update Current Price for Product | 2403 | 40.1/sec |
| GET - Product Details | 2490 | 41.5/sec |