Contents

[Testing 2](#_Toc449968769)

[Rest Client 2](#_Toc449968770)

[Product catalogue Micro services 2](#_Toc449968771)

[GET method 2](#_Toc449968772)

[Search by product Type 3](#_Toc449968773)

[Create 3](#_Toc449968774)

[Delete 4](#_Toc449968775)

[Pricing Service 4](#_Toc449968776)

[Config setup and building WAR 5](#_Toc449968777)

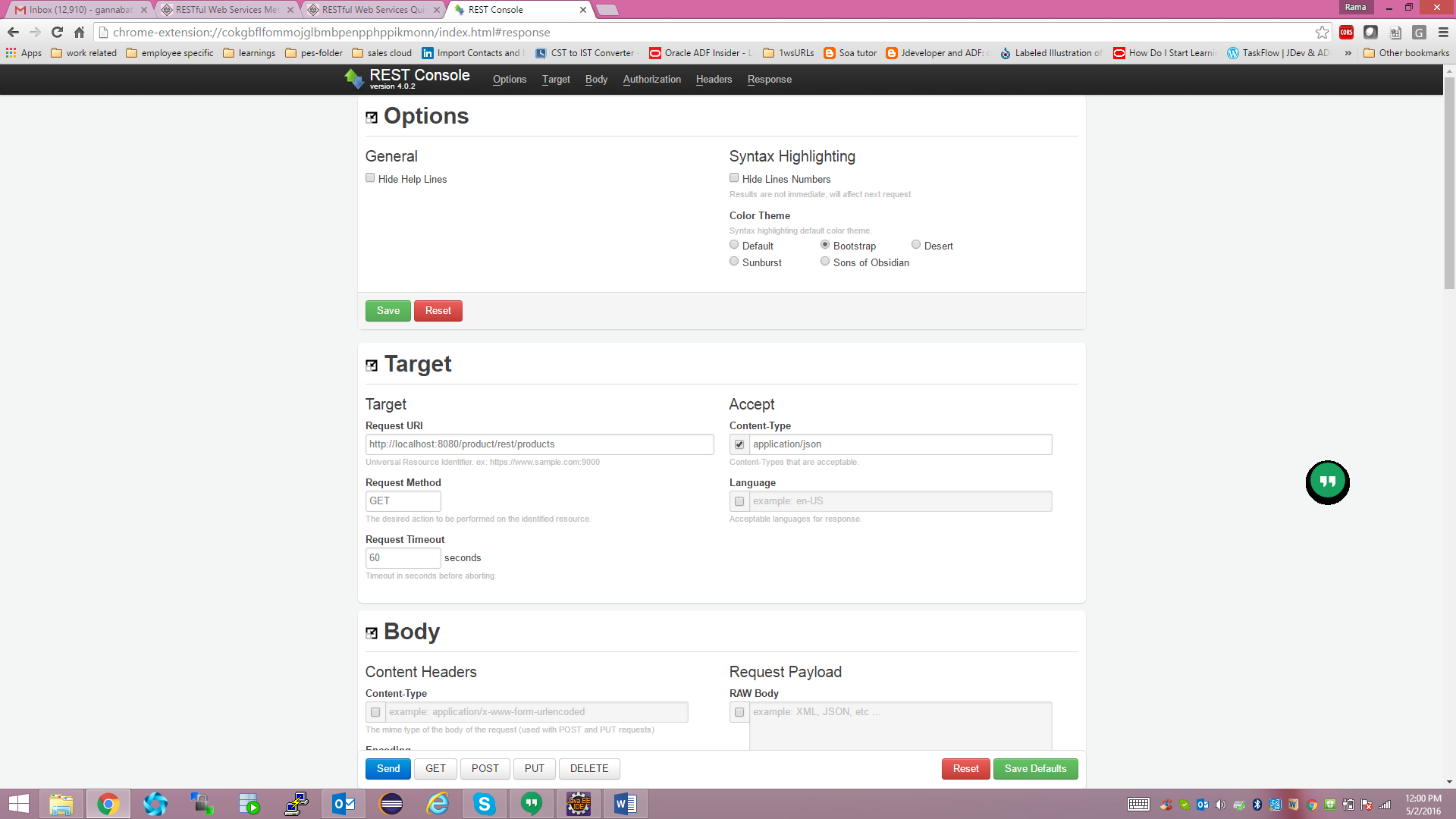
[Pricing Design 7](#_Toc449968778)

[Product service Design 8](#_Toc449968779)

# Testing

## Rest Client

Can use chrome extension



## Product catalogue Micro services

### GET method

What it does: it returns all rows in system at its simplicity. We can always add pagination features.

API: http://localhost:9080/product/rest/product

Accept type: application/json

Content type: application/json

Sample input payload: NILL

Sample output payload: list of products

1. [{
2. "productId": 1,
3. "productName": "we",
4. "productType": "y"
5. }]

### Search by product Type

What it does: it returns all rows in system at its simplicity. We can always add pagination features.

API: http://localhost:9080/product/rest/products/search?productType=y

Accept type: application/json

Content type: application/json

Sample input payload: NILL

Sample output payload: list of products

1. [{
2. "productId": 1,
3. "productName": "we",
4. "productType": "y"
5. }]

### Create

What it does: it returns all rows in system at its simplicity. We can always add pagination features.

API: http://localhost:9080/product/rest/products

Accept type: application/json

Content type: application/json

Sample input payload:

1. {
2. "productName": "rama1",
3. "productType": "x"
4. }

Sample output payload: list of products

1. {
2. "productId": 2,
3. "productName": "rama1",
4. "productType": "x"
5. }

### Delete

API: http://localhost:9080/product/rest/products/1

Accept type: application/json

Content type: application/json

Sample input payload: nill

Sample output payload: list of products

1. Success

# Pricing Service

Getting price for the given product

API : http://localhost:8080/pricing/rest/pricings/query?productId=1

Accept type: application/json

Content type: application/json

Sample input payload: nill

Sample output payload: list of products

1. {
2. "status": "SUCCESS",
3. "result": {
4. "pricingId": 1,
5. "price": 140.0,
6. "productId": 1,
7. "productName": "rama",
8. "productType": "x"
9. },
10. "message": "Pricing read successfully."
11. }

Other services are also added, but as per requirement, tested this only..

# Config setup and building WAR

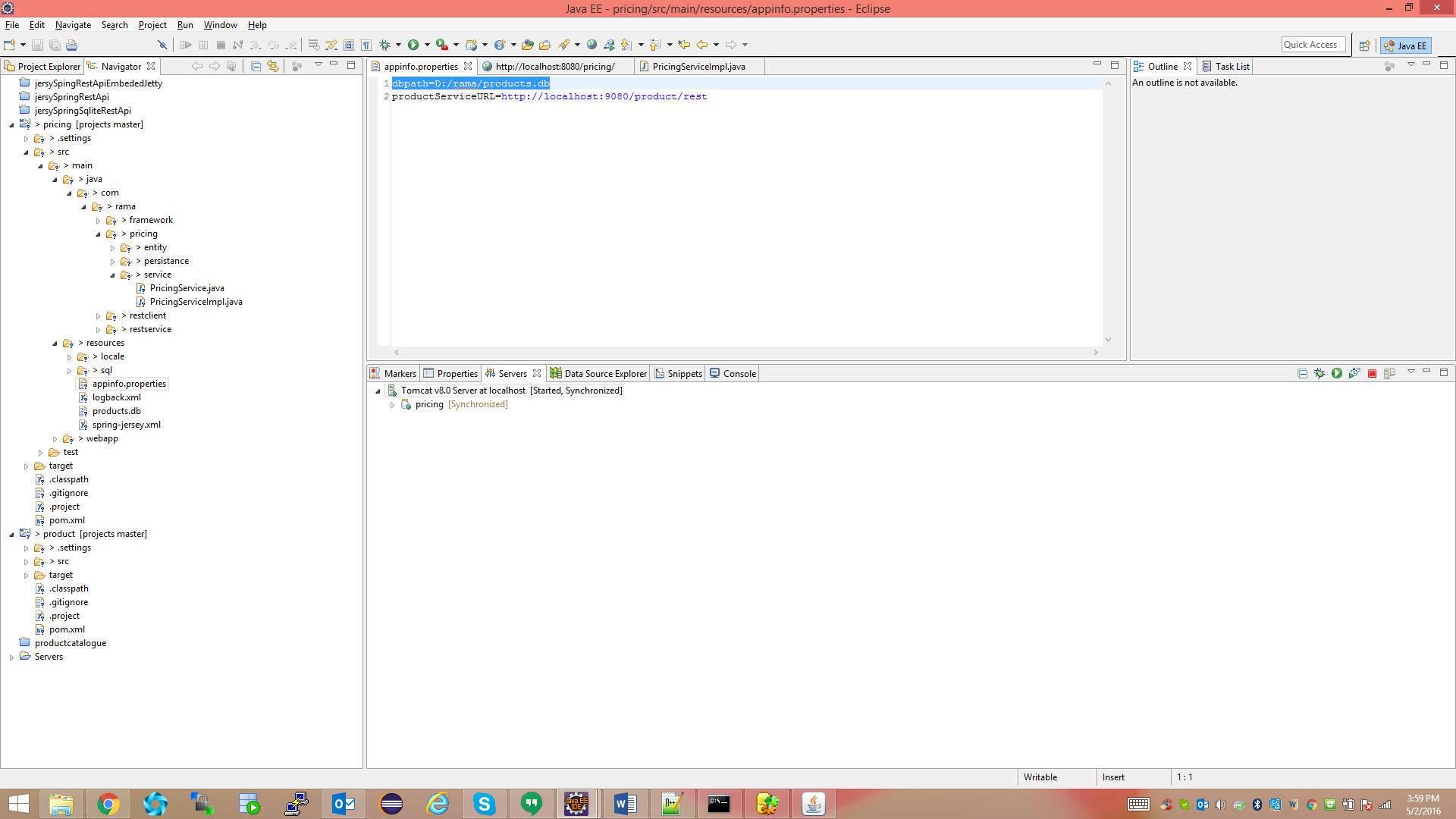
Pricing project and product project in eclipse can be imported as Maven projects.

Before building war file update the following appinfo.properties file in both projects

For pricing it should look like

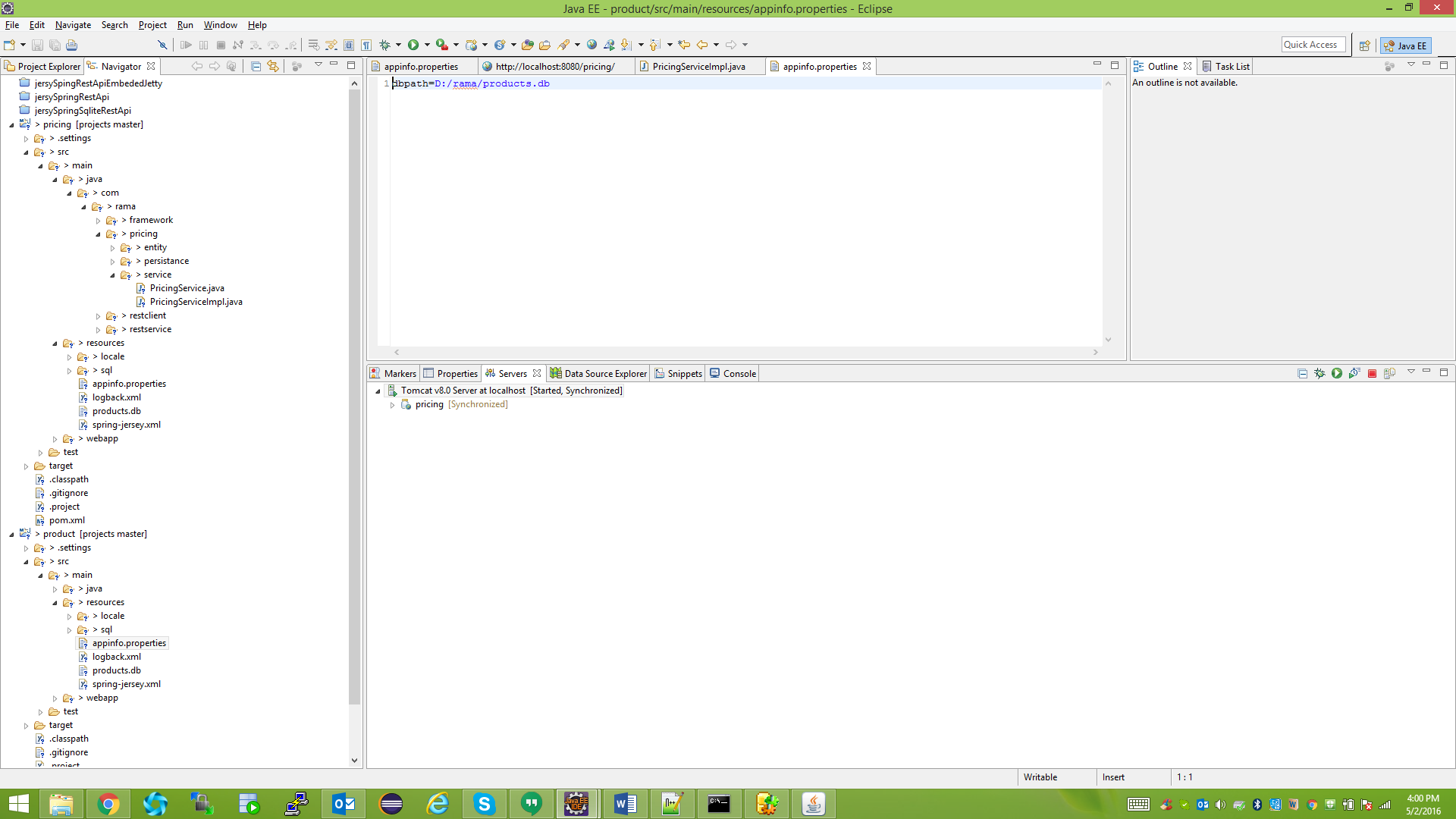
dbpath=D:/rama/products.db

productServiceURL=http://localhost:9080/product/rest



For product catalogue

dbpath=D:/rama/products.db

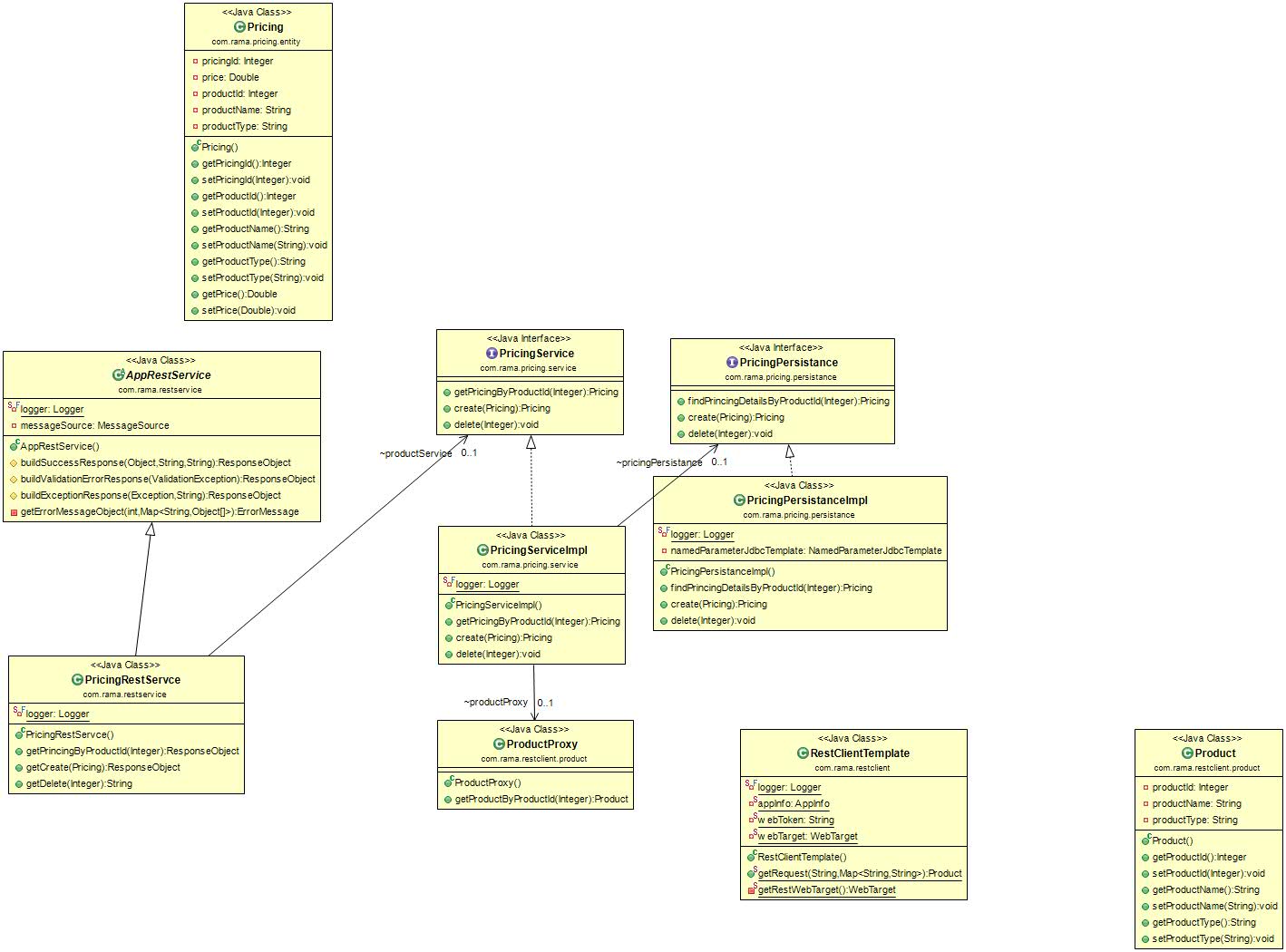


Deploy product war file on tomcat with 9080 port and pricing on 8080 port.

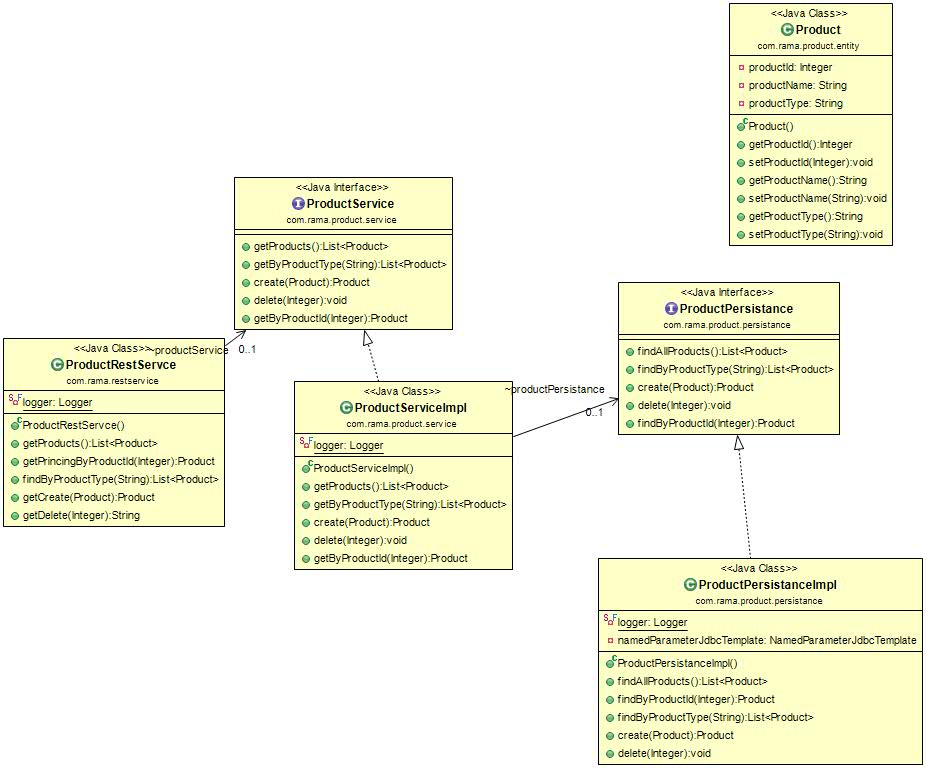
Copy the products.db file specified location in system.

Follow the testing url and payload mentioned above to verify it.

# Pricing Design



# Product service Design



For brevity,, I have implemented samples with full-fledged and rest as simple as possible.

This architecture and design is based on multiple models and principles.