

SOEN 487
Team 07 Deliverable 4
Project Documentation

Yang Shen 27159390
Mohammed Raza 26460992
Edwin Khalil Yachoui 21331442
Marco Dagostino 29587357
Tsang Chi Kit 25692636

April 20, 2018
Concordia University

Table of Contents

Project Milestone 1.1-1.2	4
A. Directory Layout	4
B. Deployment Architecture	4
C. Dependencies, Libraries and Non-Functional Requirements	4
D. User Manual and Deployment	5
Project Milestone 1.3	6
A. Directory Layout	6
B. Deployment Architecture	8
C. Dependencies, Libraries and Non-Functional Requirements	8
D. User Manual and Deployment	8
User Guide:	8
Project Milestone 2.1	10
A. Directory Layout	10
B. Deployment Architecture	12
C. Dependencies, Libraries and Non-Functional Requirements	12
D. User Manual and Deployment	12
Installation Steps:	12
Project Milestone 2.2	13
A. Directory Layout	13
B. Deployment Architecture	13
C. Dependencies, Libraries and Non-Functional Requirements	13
D. User Manual and Deployment	14
Installation Steps:	14
Project Milestone 2.3	14
A. Directory Layout	14
B. Deployment Architecture	17
C. Dependencies, Libraries and Non-Functional Requirements	17
D. User Manual and Deployment	18
Installation Steps:	18
Project Milestone 3.1	19
A. Directory Layout	19
B. Deployment Architecture	21
C. Dependencies, Libraries and Non-Functional Requirements	21
D. User Manual and Deployment	21

Installation Steps:	21
Project Milestone 3.2	22
A. Directory Layout	22
B. Deployment Architecture	24
C. Dependencies, Libraries and Non-Functional Requirements	24
D. User Manual and Deployment	24
Project Milestone 3.3	25
A. Directory Layout	25
B. Deployment Architecture	28
C. Dependencies, Libraries and Non-Functional Requirements	28
D. User Manual and Deployment	29
Project Milestone 4.1	31
Directory Layout	31
B. Deployment Architecture	31
[2]	31
C. Dependencies, Libraries and Non-Functional Requirements	31
D. User Manual and Deployment	32
Project Milestone 4.2	33
Directory Layout	33
B. Deployment Architecture	35
C. Dependencies, Libraries and Non-Functional Requirements	35
D. User Manual and Deployment	35
Petri Net	37
Sample business flow:	38
Docker	39
Maven Configurations (pom.xml)	40
Docker image building steps	41
Design for Scalability: Hadoop/MapReduce	41
Security	42
Logging Service	47
Performance Evaluation	47
REST resource only	48
SOAP method only	49
REST resource with SOAP method together	51

SC Web Application	54
Home	54
About	54
Order	55
Payment	55
Team	56
Individual Contribution	57
References	58

Project Milestone 1.1-1.2

A. Directory Layout

soen487-w18-team07

Src folder

Main folder

java/soen487 folder

Util Folder

XMLUtils Folder

DOMUtils.java

Marfcatin.java

Marfcatout.java

Neural.java

RSSPaser.java

WSDLParser.java

WSDLValuePair.java

XMLUtils.java

wsdlJaxB Folder

Defintions.java

WsdApp.java

Webapp Folder

index.jsp

B. Deployment Architecture

The user sees a jsp page where he can choose the parser he wants and the request will be sent to a servlet that will decide the proper parser depending on the choice the user gave in. It will return a web page with the parsed information.

C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0

- org.apache.httpcomponents: 4.0.2

D. User Manual and Deployment

Steps to take to start the program

1.Build and Clean the project

2.Run the project

3.Main page of the parser will appear in the browser, if not home URI is:
<http://localhost:8080/soen487-w18-team07/>

4.Choose your parser type

- RSS
- WSDL
- MARFCAT-IN
- MARFCAT-OUT
- Neural Network

5.Enter the URI of the file that it will parse

6.A Html page will appear with the parsed text

7.For testing soap service for Ex.2 (PM1.2), go to
<http://localhost:8080/soen487-w18-team07/ReaderWs?Tester>

Note: To parse the wsdl file run soen487.wsdlJaxb.WsdlApp.java (when we run locally, we get a "null")

Project Milestone 1.3

A. Directory Layout

Soen487-w18-team07 Folder

Manufacturer_client Folder

Nbproject Folder

Private Folder

private.properties

Private.xml

ant-deploy.xml

Build-impl.xml

Genfiles.properties

Jax-ws.xml

Jaxws-build.xml

Project-properties

Project.xml

Src folder

Conf Folder

java/client Folder

ManufacturerServlet.java

Web folder

WEB-INF folder

wsdl/localhost_8080/soen487-w18-team07 Folder

manufacturerService.wsdl

manufacturerService.xsd_1.xsd

jax-ws-catalog.xml

Index.jsp

paymentReceived.jsp

productInfo.jsp

purchaseOrder.jsp

Build.xml

catalog.xml

Warehouse_client Folder

Nbproject Folder

Private Folder

- private.properties
- Private.xml
- ant-deploy.xml
- Build-impl.xml
- Genfiles.properties
- Jax-ws.xml
- Jaxws-build.xml
- Project-properties
- Project.xml
- Src folder
 - Conf Folder
 - Java Folder
 - Soen487 Folder
 - Entity Folder
 - Catalog.java
 - Customer.java
 - Inventory.java
 - Item.java
 - ItemList.java
 - ItemShippingStatusList.java
 - Product.java
 - PurchaseOrder.java
 - Util/XMLUtils Folder
 - DOMUtils.java
 - XMLUtils.java
 - WarehouseServlet.java
- Web folder
 - WEB-INF folder
 - wsdl/localhost_8080/soen487-w18-team07 Folder
 - warehouseService.wsdl
 - warehouseService.xsd_1.xsd
 - Jax-ws-catalog.xml
 - web.xml
 - OrderForm.jsp
- Build.xml
- Catalog.xml

B. Deployment Architecture

Each the warehouse client and the manufacturer client have jsp pages with their appropriate servlets. For this milestone, SOAP web service architecture was used.

C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2

D. User Manual and Deployment

Installation steps:

Make Sure Clone/Download the Project under Your User Home Directory

For Windows User, check with `echo %homepath%` under Command Prompt

For Linux User, check with `echo %HOME` under Command Terminal

- Open Project With NetBean/OpenESB IDE
- Clean and Build All Project
- Server : soen487-w18-team07
- Manufacturer Client: Manufacturer_client
 - import/open the Manufacturer_client project reside within the soen* folder.
 - under project directory, select the client application
 - clean and build
- Warehouse Client: WarehouseClient
 - clean and build
- Startup Glassfish server and deploy/run all projects

User Guide:

Go to http://localhost:8080/Manufacturer_client/index.jsp? for Manufacturer client.

- Create Purchase Order:

- After successfully created an order, a file named `purchaseOrder_orderNum_unpaid.xml` will appear in server `/src/main/resources/purchaseOrder_unpaid` folder.
- Pay Order:
 - Use the order number you just created, the payment amount must be greater or equal to the quantity*unit price of the order, otherwise error message will occur.
 - Note that after successfully paying an order, the corresponding `purchaseOrder_orderNum_unpaid.xml` will be deleted from `/src/main/resources/purchaseOrder_unpaid` folder.
 - A new `purchaseOrder_orderNum_paid.xml` will appear in `/src/main/resources/purchaseOrder_paid` folder, inspecting the xml file, a child xml element `<payment>` indicates the order is paid.
- Product Information:
 - Displays detailed information about products.

Go to <http://localhost:8080/WarehouseClient/> for Warehouse client.

- A list of available items in the inventory is displayed in full details.
- Adding items to your shipping list
- Mark the items you wish to purchase and enter in the corresponding quantity value
- Enter Customer Information by filling in the corresponding input fields
- Submit your shipping order by clicking on the "Submit" button
- Once the submission has processed
- User is redirected to Warehouse client default web page
- Inventory quantity values are updated (subtracted)
 - Inventory changes will be reflected in `/src/main/resources/inventory.xml`
- A new shipping Record is created for the User containing the items they have added for shipping, with the customer information.
 - `/src/main/resources/shippingRecords/userID.xml`
- If an item's quantity value falls below a threshold of 50 we automatically replenish back the quantity amount to 200
- If an item is available in the inventory we add the item to the `shippedItemsList`
- If an item is not available, where User provides a quantity amount greater then the quantity available we add the item to the `notShippedList`
- Note, we assume that User will be providing valid inputs, therefore client side validation is not implemented.

Project Milestone 2.1

A. Directory Layout

Soen487-w18-team07 Folder

Customer_Restful Folder

Build Folder

Dist Folder

Customer_Restful.war

Nbproject Folder

Private Folder

private.properties

Private.xml

ant-deploy.xml

Build-impl.xml

Genfiles.properties

Jax-ws.xml

Jaxws-build.xml

Project-properties

Project.xml

Src Folder

Conf Folder

MANIFEST.MF

Persistence.xml

Entities Folder

Service Folder

AbstractFacade.java

ApplicationConfig.java

CustomerFacadeREST.java

DiscountCodeFacadeREST.java

MicroMarketFacadeREST.java

Customer.java

DiscountCode.java

MicroMarket.java

Web Folder

WEB-INF Folder
Index.html
Test-resbean.html
Test-resbean.js
build.xml

Customer_RestFul_Test

Dist Folder

Customer_Restfu_Testl.war

Nbproject Folder

Private Folder

private.properties
Private.xml
ant-deploy.xml
Build-impl.xml
Genfiles.properties
Jax-ws.xml
Jaxws-build.xml
Project-properties
Project.xml

Src Folder

Conf Folder

MANIFEST.MF

Web Folder

WEB-INF Folder

web.xml

Images Folder

JS Folder

Custom.js
search.js
search.html
Test-resbean.html
Test.html
Index.html
Test-resbean.js
RestClient.js
RestClientDiscount.js
build.xml

B. Deployment Architecture

This milestone uses a RESTFUL web service architecture. The Customer_Restful_Test restful app utilizes the restful web service of Customer_Restful. The user sees the the table that contains the information of customer and the discount code tables. This information is retrieve using AJAX calls. The user can then use the search to filter the data through the state attribute.

C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- gson:2.8.2

D. User Manual and Deployment

Installation Steps:

Make Sure Clone/Download the Project under Your User Home Directory Make sure to use OpenESB to open the projects

- For PM2.1
 - Open soen487-w18-team07 project, right click Customer_Restful Customer_Restful_Test, select "Open Project of Folder"
 - Make sure jdbc/sample is running and connected
 - Build and clean the Customer_Restful and Customer_Restful_Test web application
 - Deploy Customer_Restful and run Customer_Restful_Test

Project Milestone 2.2

A. Directory Layout

Soen487-w18-team07 Folder

Src Folder

Main Folder

Java Folder

TemperatureWS Folder

ApplicationConfig.java

TemperatureBean.java

TemperatureResource.java

B. Deployment Architecture

This milestone uses a RESTFUL web service architecture.

C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- gson:2.8.2

D. User Manual and Deployment

Installation Steps:

Make Sure Clone/Download the Project under Your User Home Directory Make sure to use OpenESB to open the projects

- For PM2.2
 - Clean Build and Run soen487-w18-team07 project, then follow the instruction on main page.

Project Milestone 2.3

A. Directory Layout

Soen487-w18-team07 Folder

RestfulWarehouseApp Folder

Build Folder

Entity Folder

Inventory_.java
Manufacturer_.java
Product_.java
PurchaseOrder_.java
Retailer_.java
Warehouse_.java

Dependencies Folder

Gson-2.8.0-javadoc.jar
Gson-2.8.0-sources.jar
gson-2.8.0.jar

Dist Folder

RestFulWarehouseApp.war

Nbproject Folder

Private Folder

private.properties
Private.xml
ant-deploy.xml

- Build-impl.xml
- Genfiles.properties
- Jax-ws.xml
- Jaxws-build.xml
- Project-properties
- Project.xml
- rest-build.xml

Src Folder

Conf Folder

- MANIFEST.MF
- persistence.xml

Java Folder

Client Folder

- Driver.java
- WarehouseClient.java

Entity Folder

Service Folder

- AbstractFacade.java
- ApplicationConfig.java
- InventoryFacadeREST.java
- ManufacturerFacadeREST.java
- ProductFacadeREST.java
- PurchaseOrderFacadeREST.java
- RetailerFacadeREST.java
- WarehouseFacadeREST.java
- Inventory.java
- Manufacturer.java
- Product.java
- PurchaseOrder.java
- Retailer.java
- Warehouse.java

Web Folder

WEB-INF Folder

- Glassfish-resource.xml
- Web.xml
- Index.html
- Test-resbeans.html
- Test-resbeans.js

Build.xml

Soen487-w18-team07 Folder

RetailerWebService Folder

Nbproject Folder

ant-deploy.xml
Build-impl.xml
Genfiles.properties
Jax-ws.xml
Jaxws-build.xml
Project-properties
Project.xml
Rest-build.xml

Src Folder

Conf Folder

MANIFEST.MF
persistence.xml

Java/RetailerEntity Folder

Service Folder

AbstractFacade.java
ApplicationConfig.java
InventoryFacadeREST.java
ManufacturerFacadeREST.java
ProductFacadeREST.java
PurchaseOrderFacadeREST.java
RetailerFacadeREST.java
WarehouseFacadeREST.java

Inventory.java
Manufacturer.java
Product.java
PurchaseOrder.java
Retailer.java
Warehouse.java

Web Folder

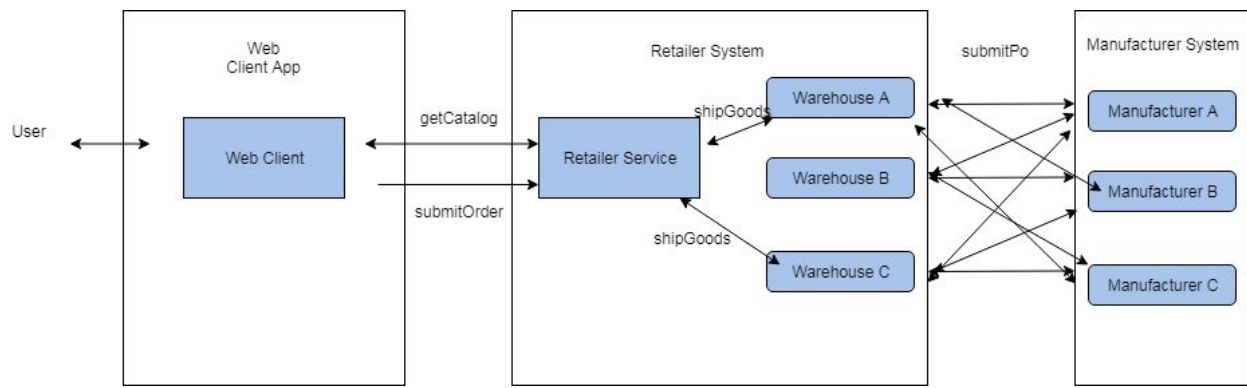
WEB-INF Folder

Glassfish-resource.xml
Web.xml
Index.html
retailerClient.html

RetailerRestClient.html
RestClient.js
Test-resbeans.html
Test-resbeans.js
build.xml

B. Deployment Architecture

Milestone 2.3 will deploy this architecture using RESTful Services as communication [1]



C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- Gson:2.8.2

D. User Manual and Deployment

Installation Steps:

Make Sure Clone/Download the Project under Your User Home Directory Make sure to use OpenESB to open the projects

- For DB setup
 - Please have local MySql server instance running
 - Open IDE, navigate to `src/main/resources/sql/` run `table_definition.sql`
 - Select `new dabtabase connection`
 - Driver -> MySQL (Connector/J driver), click Next
 - Leave Database field empty, Password -> your password for MySQL server, Test Connection, if it says "Connection Succeeded", click Next
 - Click Next
 - In the Input connection name enter
`<jdbc:mysql://localhost:3306/soen487_w18_team07?zeroDateTimeBehavior=convertToNull [root on Default schema]>`
 - Click Finish
 - Navigate to `src/main/java/soen487/util/DBUtils/DbInitiator.java` change the PASSWORD attribute to your mysql password.
 - Run the file, it will generate random data into our db.
- For PM2.3
 - Clean&Build and Deploy&Run `RestfulWarehouseApp` and `RetailerWebService` project.
 - For `RetailerWebService` please run
`/RestfulWarehouseApp/src/java/soen487/client/Driver.java` for interactive console.

Project Milestone 3.1

A. Directory Layout

Soen487-w18-team07 Folder

HelloWorld Folder

Build Folder

META-INF Folder

Catalog.xml

jbi.xml

HelloWorld.wsdl

SEDeployment.jar

helloWorld.bpel

Nbproject Folder

Private Folder

private.properties

Private.xml

Build-impl.xml

Genfiles.properties

Project-properties

Project.xml

Src Folder

HelloWorld.wsdl

helloWorld.bpel

Build.xml

catalog.xml

Soen487-w18-team07 Folder

HelloWorldClientApp Folder

Build Folder

META-INF Folder

Catalog.xml

jbi.xml

HelloWorld.wsdl

SEDeployment.jar

sun-http-binding.jar

Dist Folder

HelloWorldClientApp.zip

Nbproject Folder

Private Folder

private.properties

Private.xml

Build-impl.xml

Genfiles.properties

Project-properties

Project.xml

Src Folder

Conf Folder

AssemblyInformation.xml

BindingComponentInformation.xml

ComponentInformation.xml

HelloWorldClientApp.casa

jbiServiceUnits Folder

HelloWorldClientApp.wsdl

catalog.xml

Jbiosa Folder

HelloWorldClientApp.wsdl

Test Folder

TestCase1 Folder

Concurrent.properties

Input.xml

output.xml

Results Folder

TestCase1 Folder

Actual_20180327213019_F.xml

TEST-org.netbeans.modules.compapp.catd.ConfiguredTest.xml

All-tests.properties

selected-tests.properties

Build.xml

Catalog.xml

Instructions.html

B. Deployment Architecture

A Bpel process that implements a simple hello bpel business process. The process accepts the input "AAA" and will return the message "Hello AAA".

C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- gson:2.8.2

D. User Manual and Deployment

Installation Steps:

Make Sure Clone/Download the Project under Your User Home Directory Make sure to use OpenESB to open the projects

- HelloWorld BPEL project (PM3.1)
 - run OpenESB Standalone server
 - clean and build bepel project "HelloWorld"
 - clean and build Composite Application "HelloWorldClientApp"
 - deploy "HelloWorldClientApp"
 - select your OpenESB Standalone server
 - run "TestCase1" to test input/output
 - pass argument in "input.xml" into "part1" element
 - make sure "output.xml" is clean

Project Milestone 3.2

A. Directory Layout

Soen487-w18-team07 Folder

WeatherRESTfulBpel Folder

Nbproject Folder

Build-impl.xml
Genfiles.properties
Project.properties
project.xml

Src Folder

WeatherBpel.bpel
WeatherRequestor.wsdl
WeatherRequestorWrapper.wsdl
WeatherServiceProvider.wsdl

Build.xml

Catalog.xml

WeatherRESTCA Folder

Nbproject Folder

Build-impl.xml
Genfiles.properties
Project.properties
project.xml

Src Folder

Conf Folder

AssemblyInformation.xml
BindingComponentInformation.xml
ComponentInformation.xml
WeatherRESTCA.casa

Jbiosa Folder

WeatherRESTCA.wsdl

Test Folder

TestCase1 Folder

Concurrent.properties
Input.xml
output.xml
selected-tests.properties
Build.xml
Catalog.xml

XMLReaderApp Folder

Nbproject Folder

Build-impl.xml
Genfiles.properties
Project.properties
project.xml

Src Folder

AssemblyInformation.xml
BindingComponentInformation.xml
ComponentInformation.xml
XMLReaderApp.casa

Test Folder

TestCase1 Folder

Concurrent.properties
Input.xml
output.xml
all-tests.properties
selected-tests.properties
Build.xml
Catalog.xml

XMLReaderBpel Folder

Nbproject Folder

Build-impl.xml
Genfiles.properties
Project.properties
project.xml

Src Folder

XMLReader.wsdl
XMLReaderBpel.wsdl

XMLReaderRequestor.wsdl
XMLReaderWrapper.wsdl
Build.xml
catalog.xml

B. Deployment Architecture

Bpel process that invokes the XMLReaders from milestone 1.2. This process also invokes the Weather Service from milestone 2.2.

C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- gson:2.8.2

D. User Manual and Deployment

- WeatherRESTfulBPEL project (PM3.2)
 - Import the following projects in OpenESB with OpenESB Standalone server running.
 - make sure to run GlassFish server prior to running OpenESB Standalone * XMLReaderApp * XMLReaderBPEL * WeatherRESTCA * WeatherRESTfulBPEL
 - Clean and Build the XMLReaderApp and XMLReader.
 - Run the projects and the test case under each project.
 - For XMLReaderApp, change type (rss, marfcats-in/out, wsdl) and url
 - For WeatherRESTCA, change the city ID. See http://openweathermap.org/help/city_list.txt

Project Milestone 3.3

A. Directory Layout

Soen487-w18-team07 Folder

ScBepel Folder

Nbproject Folder

Build-impl.xml

Genfiles.properties

Project.properties

project.xml

Src Folder

OrderProcess.bpel

bepelManufacturerServiceWSDL.wsdl

bepelManufacturerServiceWSDLWrapper.wsdl

bepelWarehouseServiceWSDL.wsdl

bepelWarehouseServiceWSDLWrapper.wsdl

Product.wsdl

productWrapper.wsdl

Build.xml

Catalog.xml

ScApplication Folder

Nbproject Folder

Build-impl.xml

Genfiles.properties

Project.properties

project.xml

Src Folder

Conf Folder

AssemblyInformation.xml

BindingComponentInformation.xml

ComponentInformation.xml

ScApplication.casa

Jbiosa Folder

ScApplication.wsdl

Test Folder

TestCase1 Folder

Concurrent.properties

Input.xml

output.xml

selected-tests.properties

Build.xml

Catalog.xml

ScClient Folder

Nbproject Folder

ant-deploy.xml

Build-impl.xml

Genfiles.properties

Project.properties

project.xml

Src Folder

MANIFEST.MF

Web Folder

index.html

build.xml

SRC Folder

Main Folder

Java Folder

Config Folder

ApplicationConfig.java

Soen487 Folder

Client Folder

CatalogServlet.java

ReaderClientServlet.java

WarehouseClientServlet.java

Entity Folder

Catalog.java

Customer.java

Inventory.java

InventorySOAP.java

Item.java

ItemList.java
ItemShippingStatusList.java
Manufacturer.java
Product.java
PurchaseOrder.java
Retailer.java
Warehouse.java

Service Folder

ManufacturerService.java
ReaderWS.java
RetailerService.java
WarehouseService.java
bepelManufacturerService.java
bepelWarehouseService.java

Resource Folder

META-INF Folder

Persistence.xml

PurchaseOrderPaid Folder

PurchaseOrderUnpaid Folder

SchemaTemplates Folder

Fault.java
ObjectFactory.java
Package-info.java

ShippingRecords Folder

SQL Folder

wSDL Templates Folder

Fault.xsd
Catalog.xml
faultSample.wsdl
Inventory.xml
log4j.properties

Webapp Folder

WEB-INF Folder

web.xml

CSS Folder

catalog.css

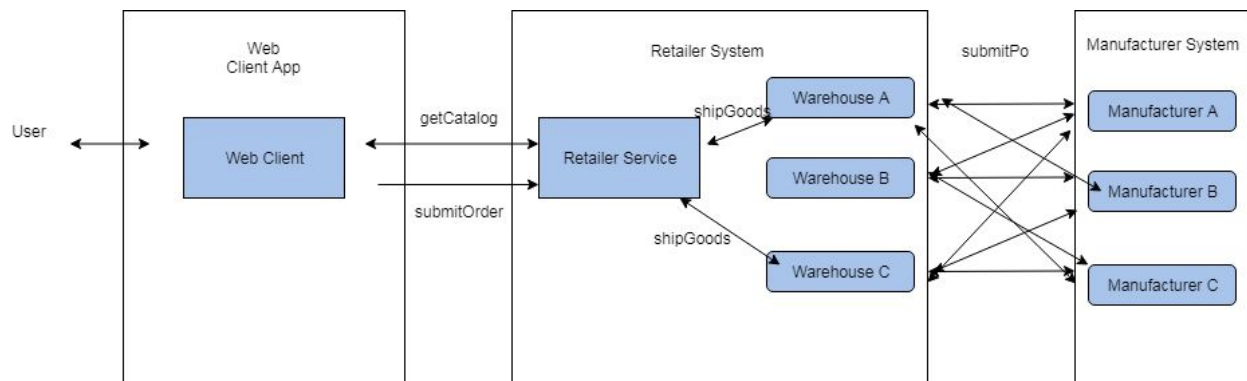
JS Folder

Catalog.js
Catalog.jsp

Index.html
Index.jsp
orderForm.jsp
serverIndex.jsp
Test-resbeans.css
Test-resbeans.js
Test-service.html
jax-ws-catalog.xml

B. Deployment Architecture

Milestone 3.3 will deploy this Web Services architecture [1]



C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- gson:2.8.2

D. User Manual and Deployment

Make Sure Clone/Download the Project under Your User Home Directory

Make sure to use OpenESB to open the projects

In order to run the Web application and browse the retailer's catalog:

- Clean and Build soen487-w18-team07 project
- Start GlassFish server. Make sure GlassFish server is running properly
- Run the soen487-w18-team07 project
- A web page will load, navigate to "View Catalog"
(URL: <http://localhost:8080/soen487-w18-team07/Catalog.jsp>)
 - Form is validated before submitting a new order
 - Client cannot submit an empty order with quantity below or above the inventory availability.

In order to run the Bpel Business process:

- Run GlassFish and openESB standalone servers simultaneously, to avoid port conflict, make sure to run both servers on different ports
- Clean and build maven project "soen487-w18-team07" then run the project
- Click on "View Catalog" or go to
["http://localhost:8080/soen487-w18-team07/Catalog.jsp"](http://localhost:8080/soen487-w18-team07/Catalog.jsp)
 - Access catalog products via retailer "RESTFUL" service (test uri:
<http://localhost:8080/soen487-w18-team07/webresources/retailer>) outputs
json data
- Enter product quantity for the products you wish to purchase
- Validation: input value must be greater than zero and less than or equal to 100
- Validation: must at least have input value for product quantity input field
- Enter customer information
- Click on "submit" button to process your order once pre-process is completed
- Will be redirected to new page where we display selected items grouped based on their shipping status
- If items are purchased successfully, a xml record of the order is created in
"resources/shippingRecords/{customerID}"
- In the backend "replenish" function is called to check and produce items if item quantity falls below 50.
- "catalog.xml" is updated - where we keep a list of available products
- Items not shipped based on quantity value exceeding available inventory quantity

- Click on "Go back to catalog" button to continue shopping as a new customer

BPEL BUSINESS PROCESS

- Clean and build "ScBEPEL"
 - We have imported WarehouseService and ManufacturerService WSDL files
 - Explanation of "OrderProcess.bpel"
 - "ProductRequestor" is acting as the retailer client
 - Input: productType and quantity
 - Output: productStatus
1. Receive the clients' input values and invoke the "shipGood" method for the first warehouse service.
 2. Check if the warehouse has enough inventory quantity for selected product.
 - a. If True, we continue the process.
 - b. If False, we invoke the next warehouse service and repeat the process until we run out of our warehouse services.
 3. Send back a reply to retailer client with a product status "TRUE" suggesting that the product is available for purchase or "FALSE" suggesting that the product is not available for purchase.
 4. Asynchronously invoke three manufacturer services to process the purchase order, where we do not wait for a response to proceed.
- To test, clean and build "ScApplication", then deploy
 1. Under "Test/TestCase1" Open "input.xml" and enter values for "productType" and "quantity - you may see "catalog.xml" to enter valid input values located in "/resources"
 2. Make sure output.xml is empty. Then run the test case. First run will fail, re-run the test case and check for "productStatus" value
 - a. "TRUE" suggests that product quantity is available
 - b. "FALSE" suggests that none of the warehouses have enough inventory for provided "productType"
 3. A purchase order is created in "resources/purchaseOrder_unpaid/{#}"

Project Milestone 4.1

Simple Hadoop/HDFS/MapReduce exercise

A. Directory Layout

4.1-Hadoop

Mapper.py

Reducer.py

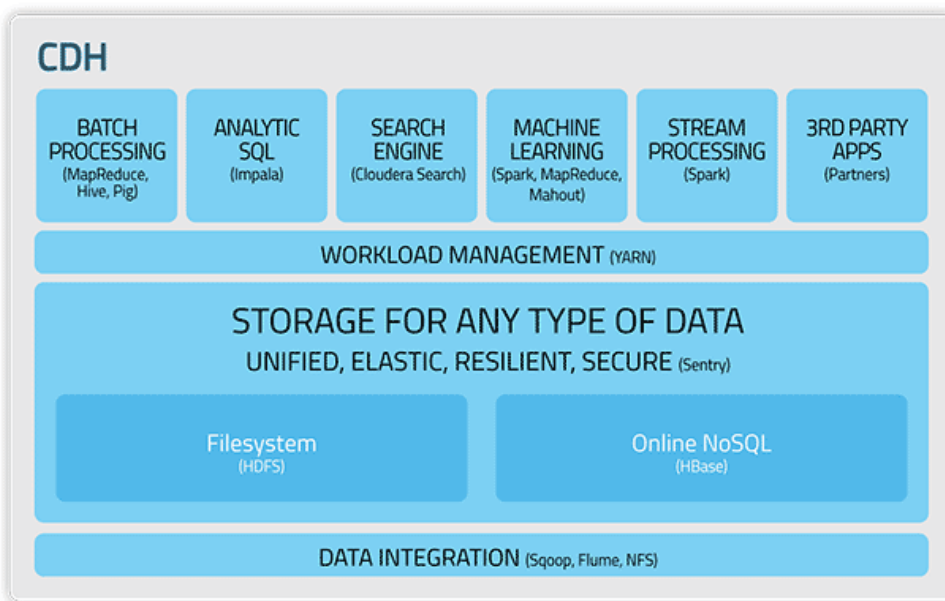
outputMapper.txt

outputMapReducer.txt

A_Tale_Of_Two_Cities.txt

B. Deployment Architecture

Here is the architecture that comes with Cloudera Hadoop



[2]

C. Dependencies, Libraries and Non-Functional Requirements

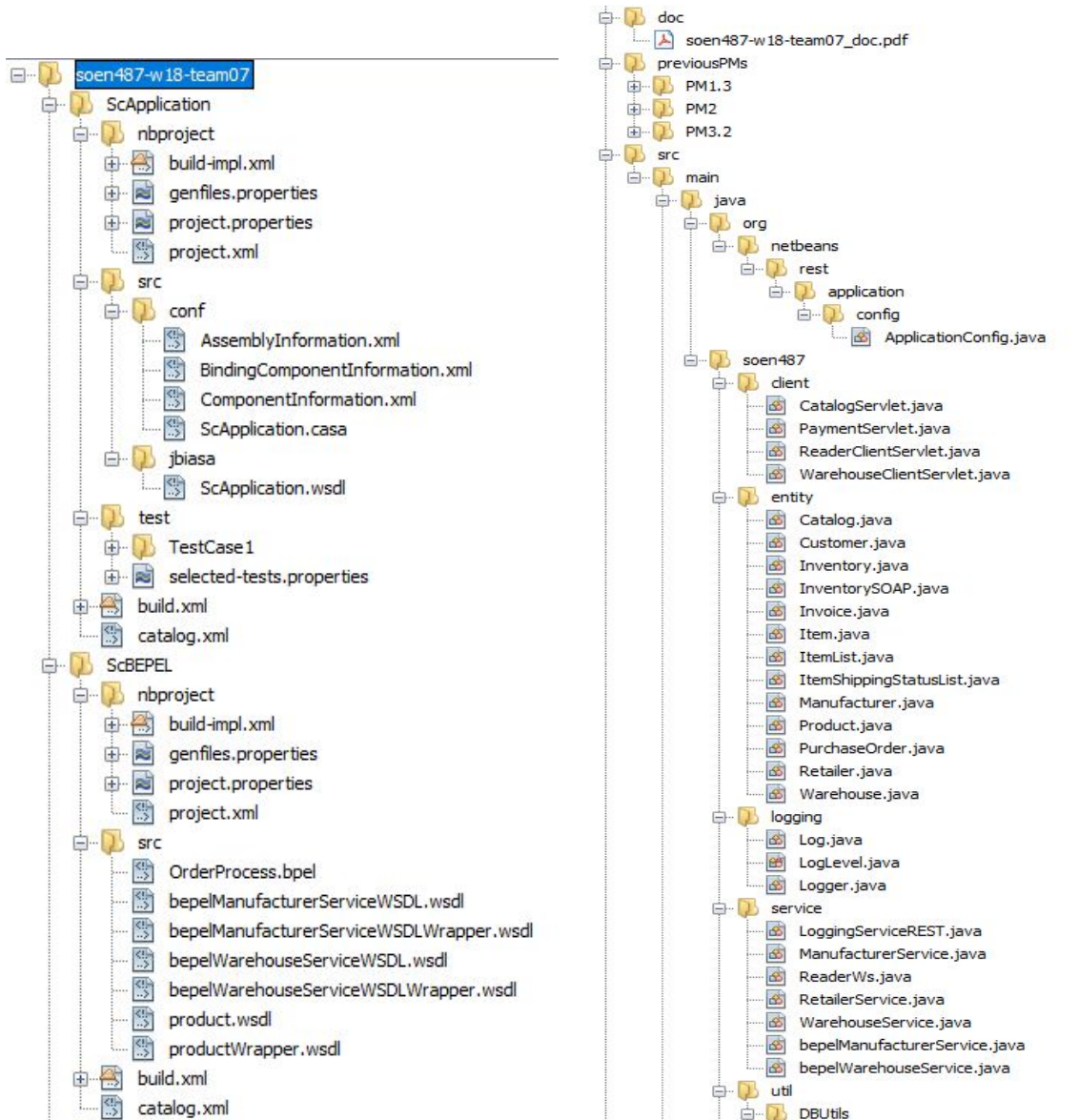
- Cloudera Hadoop
- Python 2.7 and higher

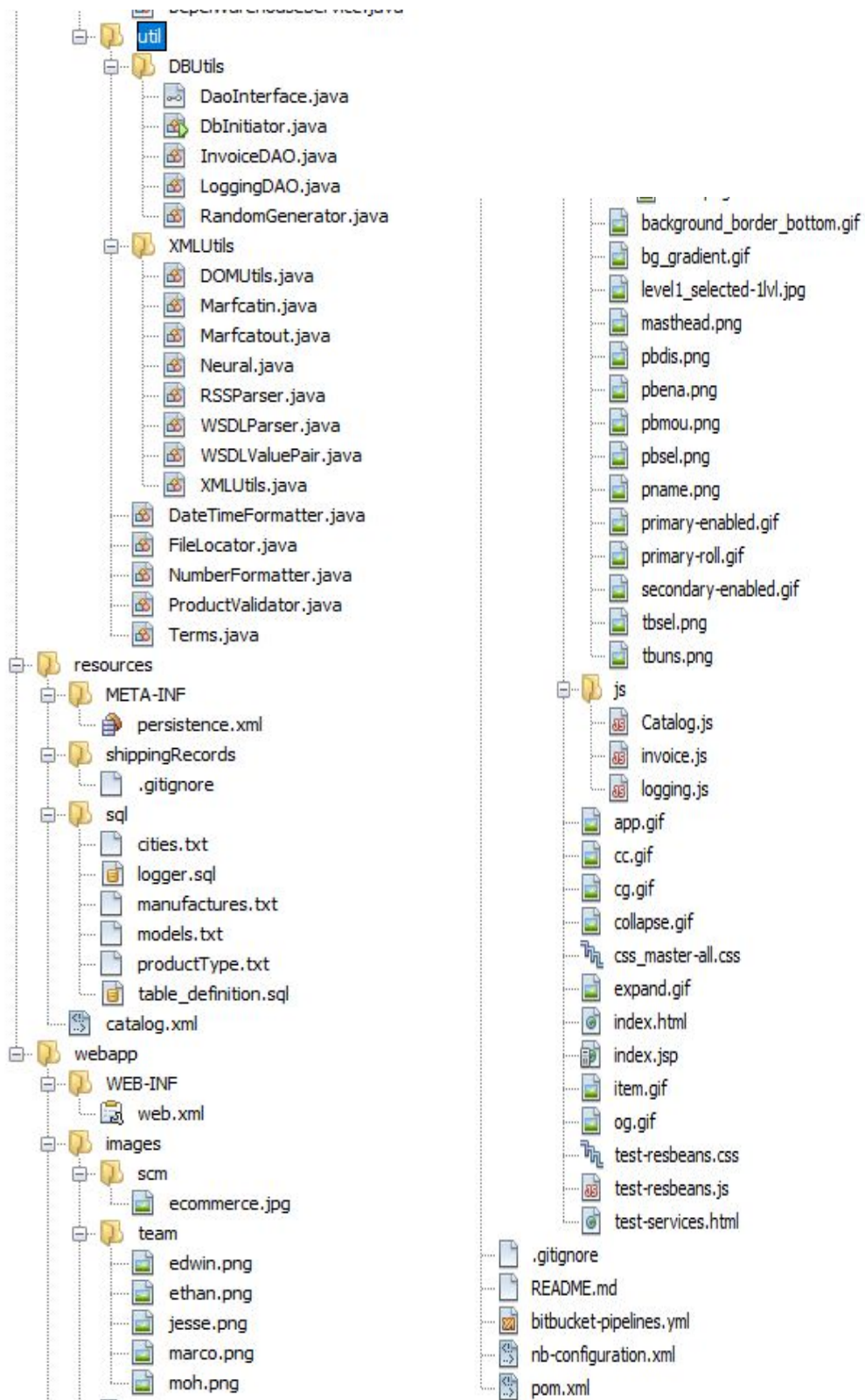
D. User Manual and Deployment

- Use Virtualbox with cloudera
- Firstly copy the Files Mapper.py, Reducer.py and A_Tale_Of_Two_Cities.txt into your own directory of your choice
- Secondly, HDFS has to be installed on the machine
- Make sure the two python files are executable if not use this command in the terminal
 - `chmod +x mapper.py`
 - `chmod +x reducer.py`
- Once HDFS is installed, the following commands have to be followed
 - Create a directory for the tutorial in hdfs:
> `hadoop fs -mkdir /user/cloudera/mydata`
 - Copy test data from local_files to hdfs:
> `hadoop fs -copyFromLocal yourDirectory/A_Tale_Of_Two_Cities.txt /users/cloudera/mydata/`
 - Check file is copied successfully in hdfs:
> `hadoop fs -ls /users/cloudera/mydata/`
 - Get all the content of the file from hdfs:
> `hadoop fs -cat /user/cloudera/mydata/A_Tale_Of_Two_Cities.txt`
- Command to run the mapper only and get an output file directly from the terminal
 - `Hadoop fs -cat /user/cloudera/mydata/A_Tale_Of_Two_Cities.txt | /home/cloudera/script/./mapper.py | sort >outputMapper.txt`
- Command to run the reducer and get an output file directly from the terminal
 - `Hadoop fs -cat /user/cloudera/mydata/A_Tale_Of_Two_Cities.txt | /home/cloudera/script/./mapper.py | sort | /home/cloudera/script/./reducer.py | sort -nk2 > outputMapperReducer.txt`

Project Milestone 4.2

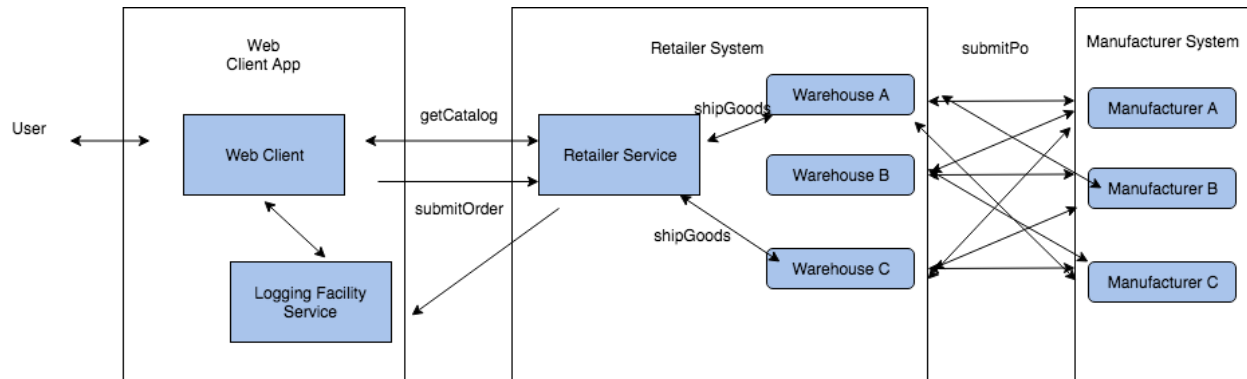
A. Directory Layout





B. Deployment Architecture

Milestone 4.2 will deploy this Web Services architecture [1]



C. Dependencies, Libraries and Non-Functional Requirements

Maven Dependencies:

- log4j :1.2.12
- org.glassfish.metro : 2.3
- org.eclipse.persistence:2.5.2
- javax:7.0
- org.apache.httpcomponents: 4.0.2
- mysql:5.1.21
- com.sun.jersey:1.14

External Library:

- gson:2.8.2

D. User Manual and Deployment

Installation Steps:

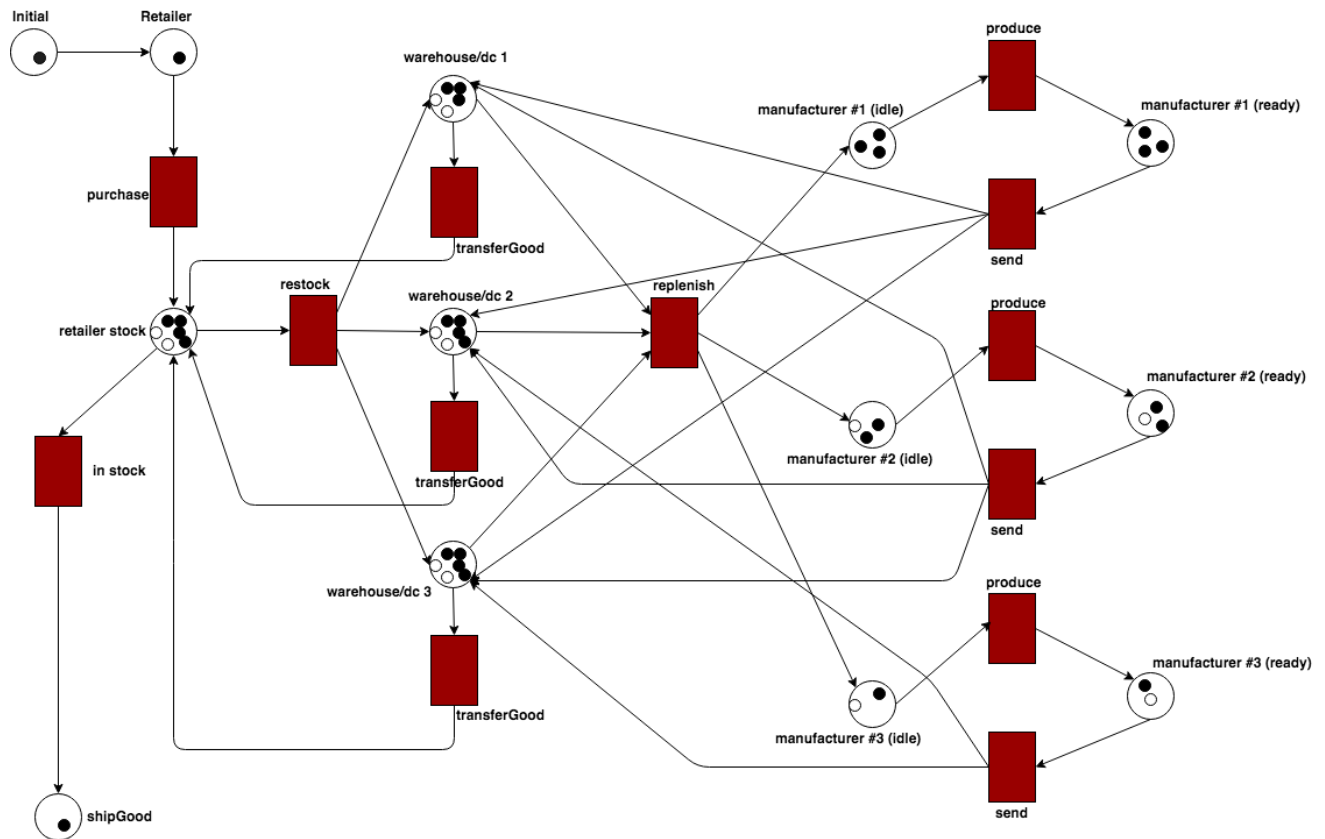
!! IMPORTANT !!

Make Sure Clone/Download the Project under Your User Home Directory MySQL server is required for this project, make sure it runs properly

- For DB setup
 - Please have local MySQL server instance running
 - open MySQL in terminal and create a new database
 - create database soen487_w18_team07_logging;
 - Open IDE (OpenESB)

- Navigate to `src/main/resources/sql/` run `logger.sql`, select new database connection
- Driver -> MySQL (Connector/J driver), click next
- Database -> `soen487_w18_team07_logging`, Password -> your password for MySQL server
- Test Connection, if it says "Connection Succeeded", click Next
- Click Next -> Finish
- Navigate to `src/main/resources/db.properties` change the 'USER' and PASSWORD attributes to your mysql user and password.
- To run Project
 - Run GlassFish server and clean build the project.
 - Clean & Build the project.
 - Run project, url should be <http://localhost:8080/soen487-w18-team07/>

Petri Net



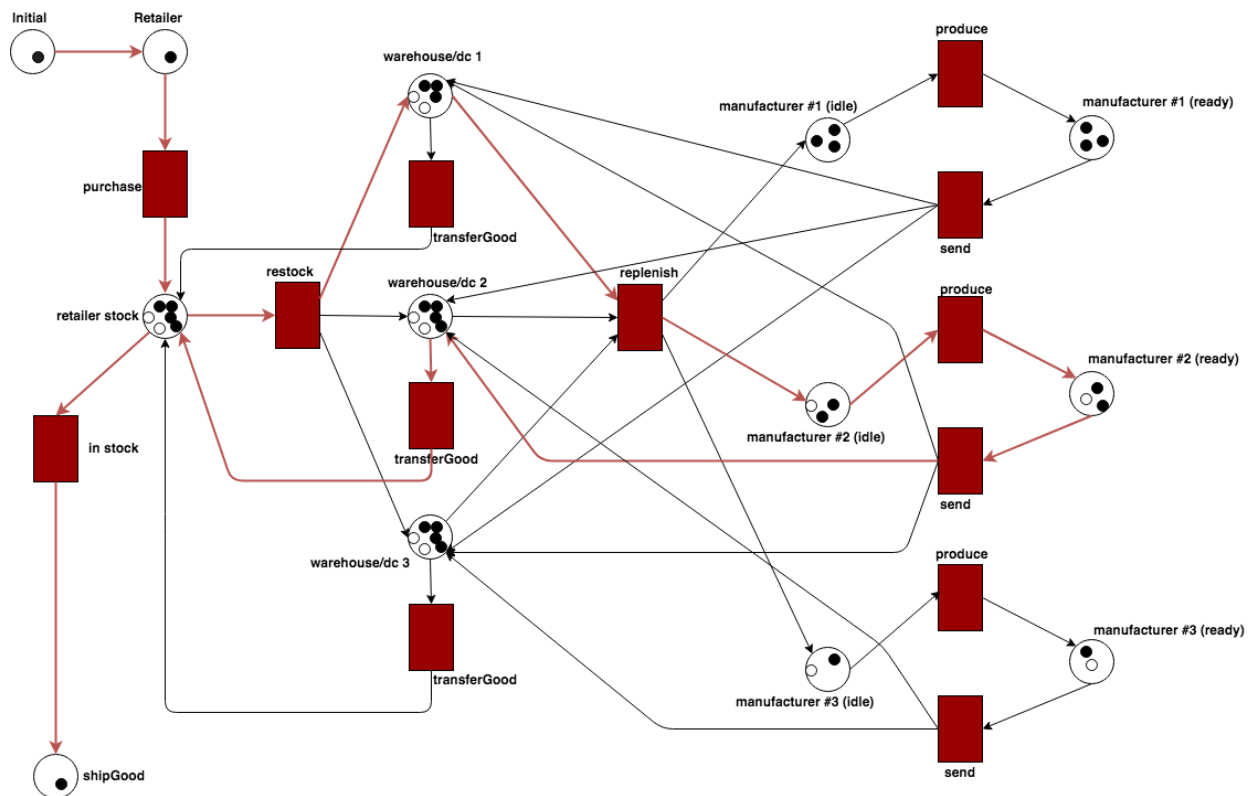
Petri Net Specification

- Commodity/Product
- Place: Customer/Retailer/Manufacturer
State: shipGood()
- Place where contains multiple product: retailer inventory/warehouse
State: manufacturer idling/manufacturer ready to send
- Action

Sample business flow:

- Customer arrives at retailer, placed an order on a product
- If retailer's have the product in stock, ship good to customer
- If retailer out of stock, restock from one of the three warehouses
- Warehouse transfer good to retailer.
- If warehouse have a commodity below certain threshold, then place an purchase order to corresponding manufacturer.
- Upon receiving purchase order from warehouse, manufacturer produce goods, changing idle state to ready.
- Manufacturer send product to warehouse.

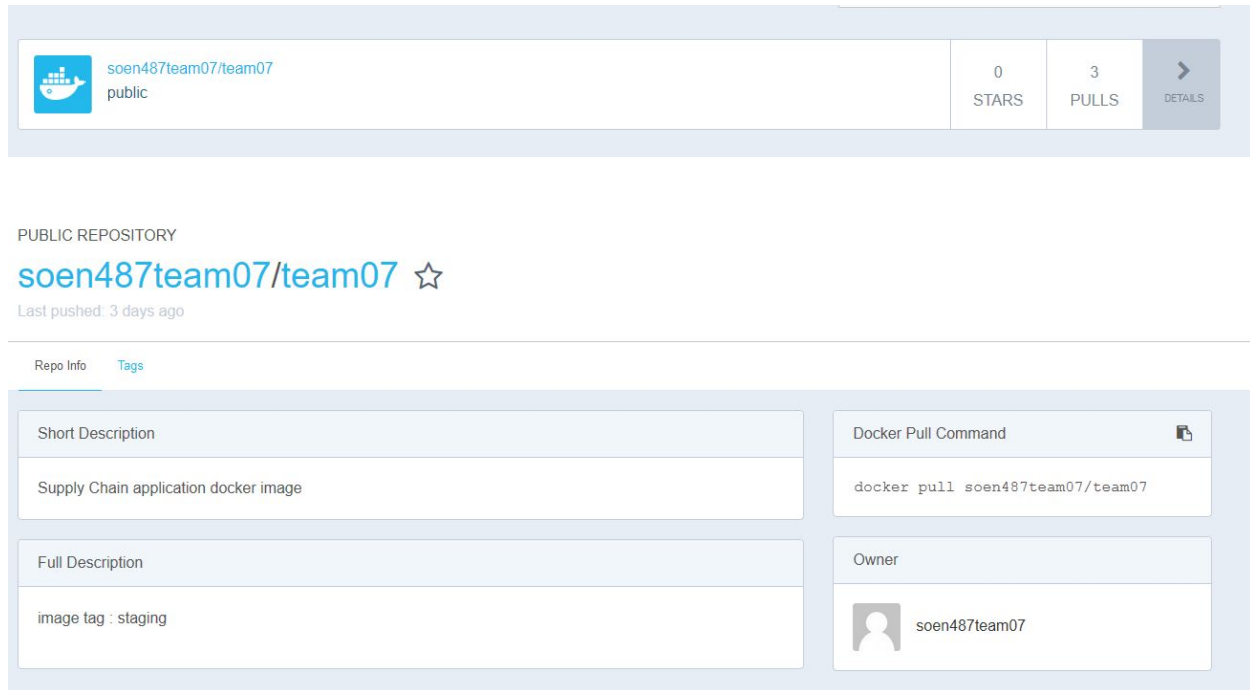
Sample flow:



Docker

- This docker implementation pertains to **PM 3.3** implementation.

Docker Repository: <https://hub.docker.com/r/soen487team07/team07/>



PUBLIC REPOSITORY

soen487team07/team07 ☆

Last pushed: 3 days ago

Repo Info Tags

Short Description

Supply Chain application docker image

Full Description

image tag : staging

Docker Pull Command

```
docker pull soen487team07/team07
```

Owner

soen487team07

- Make sure to have docker installed and running
- Make sure to have installed a docker image of glassfish
 - docker pull glassfish
- Fetch SC docker image
 - docker pull soen487team07/team07:staging
- Mount SC image with glassfish to a docker container
 - docker run -d -ti -v soen487team07/team07 -p 4848:4848 -p 8080:8080 glassfish
- Check for running containers
 - docker ps

```
PS C:\Users\Moh\soen487-w18-team07> docker run -d -ti -v soen487team07/team07 -p 4848:4848 -p 8080:8080 glassfish
028a676cfb43ff400ea7b024e3ef6588b585698c2399976b00baa7d98511cbc
PS C:\Users\Moh\soen487-w18-team07> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
028a676cfb44   glassfish "/bin/sh -c 'asadmin..." 15 seconds ago Up 14 seconds 0.0.0.0:4848->4848/tcp, 0.0.0.0:8080->8080/tcp, 8181/tcp  zealous_yalow
```

- Run localhost in your browser with port 8080.

Maven Configurations (pom.xml)

- We are utilizing Shopify docker plugin for maven type projects.

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.4.13</version>
  <configuration>
    <imageName>sc-docker</imageName>
    <baseImage>java</baseImage>
    <entryPoint>["java", "-jar", "/${project.build.finalName}.war"]</entryPoint>
    <!-- copy the service's jar file from target into the root directory of the image -->
    <resources>
      <resource>
        <targetPath>/</targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.build.finalName}.war</include>
      </resource>
    </resources>
    <forceTags>true</forceTags>
    <imageTags>
      <imageTag>${project.version}</imageTag>
      <imageTag>sc-docker</imageTag>
    </imageTags>
  </configuration>
  <executions>
    <execution>
      <id>build-image</id>
      <phase>package</phase>
      <goals>
        <goal>build</goal>
      </goals>
    </execution>
    <execution>
      <id>tag-image</id>
      <phase>package</phase>
      <goals>
        <goal>tag</goal>
      </goals>
      <configuration>
        <image>sc-docker:${project.version}</image>
        <newName>registry.example.com/sc-docker:${project.version}</newName>
      </configuration>
    </execution>
    <execution>
      <id>push-image</id>
      <phase>deploy</phase>
      <goals>
        <goal>push</goal>
      </goals>
      <configuration>
        <imageName>registry.example.com/sc-docker:${project.version}</imageName>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Docker image building steps

- Install maven
- Inside the soen487-w18-team07 project directory run the following commands
 - mvn clean build package docker:build
 - docker tag <imageName> username/repository:<tag>
 - docker push username/repository:tag
 - docker images

```
PS C:\Users\Moh\soen487-w18-team07> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
soen487team07/team07	staging	585170a37038	2 days ago	647MB
java	latest	d23bdf5b1b1b	15 months ago	643MB
glassfish	latest	7d04a1452d00	19 months ago	773MB

Design for Scalability: Hadoop/MapReduce

Hadoop is being widely used in the supply chain domain, because it is best suited for unstructured and semi-structured data. With the use Hadoop Distributed File System (HDFS) and MapReduce we can work with immense volumes of data and scale linearly from just a few to thousands of commodity computers. For example, we can breakdown the web application workflow into sub components such as data management and programming.

For our supply chain application, we manage data with the use of relational database and XML data object storage. In order to support more users, a bigger workload, or more data, we can only scale up to a certain limit; since scaling performance beyond the physical boundaries is way more complex. For example, most vendors only provide solutions that are complex and that don't scale linearly. Therefore, managing extreme volumes of data with relational database technologies remains a challenge. In addition, relational database can inject data rapidly; however, they are not architected to rapidly turn business logic into information and flow it through. Therefore, with the use of MapReduce we can process data quickly to provide rapid response to emerging business events. Furthermore, relational database deals with highly structured numerical data. They are not intended to handle text, images, or other data types that have little or no structure to provide context. As a result, since our supply chain application processes communication over XML and JSON data types, it is required to use Hadoop/HDFS. In terms of cost, the Hadoop approaches to data management are substantially less than the cost of relational database approaches.

With an increase demand of data analytics, we can utilize the Hadoop framework to store and process large quantity of data. Big data management has tremendous implications for our supply chain application. For example, we can aggregate, filter, log, and analyze internal data, as well as external consumer and market data. To add, we can use the insights generated to optimize all levels of the supply chain infrastructure. Moreover, by having integrated big data

management into our supply chain application, we will have a decisive competitive advantage over those that do not. To continue, by strengthening our supply chain application with the use of Hadoop, we can get products and services for consumers quickly and efficiently. In addition, we can include real-time analytics to better facilitate our warehousing and manufacturing services. For example, we can develop mathematical models that forecast a wide range of variables such as the lookup speed for a specific product across all warehouses. Furthermore, we can access archived data such as logs to better optimize the business process and for future development.

Thus, with regards to storage, analysis, and logging, our supply chain application is suitable and applicable to run over Hadoop/HDFS. We have seen that Hadoop provides big data advantages in terms of volume, velocity, and variety of data. In addition, as our supply chain application gets bigger, more complex, and demand faster responses time, it is mutually agreed that the use of Hadoop technology is essential for our application.

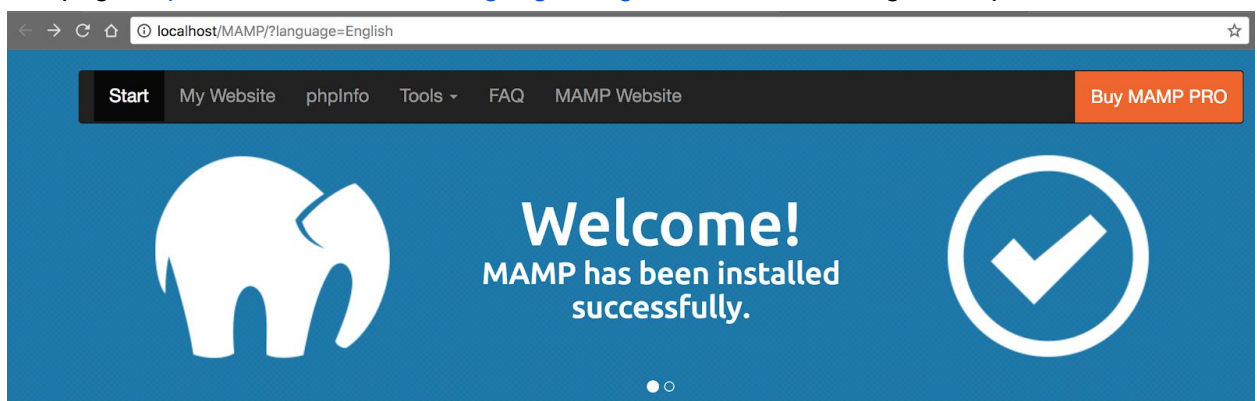
Security

HTTPS allows data to be encrypted while transferred from the web browser to the web server and vice versa. The data transmitted is encrypted by a valid certificate that is issued by a valid certificate issuing authority.

In order to to setup your local web server to run over HTTPS we will create a self signed certificate. Please follow the steps below to enable SSL.

Note: The instructions provided are based on macOS work environment.

- Download and Install MAMP as your webserver <https://www.mamp.info/en/downloads/>
- Run MAMP and click on Start Servers (Apache Server & MySQL should turn green)
- The page <http://localhost/MAMP/?language=English> will load with regular http



- Navigate to your user folder. Go to finder, Go→ Go to Folder → enter “~”

```
Edwins-MBP:~ edwinyachoui$  
/Users/edwinyachoui
```

- Create an empty folder called “ssl”. Terminal command “*mkdir ssl*”

```
Edwins-MBP:~ edwinyachoui$ mkdir ssl
```
- Cd into the newly created ssl folder.

```
Edwins-MBP:~ edwinyachoui$ cd ssl
```
- Create 2 empty files. Terminal command “*touch server.csr.cnf*” & “*touch v3.ext*”

```
Edwins-MBP:ssl edwinyachoui$ touch server.csr.cnf
```

```
Edwins-MBP:ssl edwinyachoui$ touch v3.ext
```
- Using a text editor, copy paste the following into server.csr.cnf

```
[req]
default_bits= 2048
prompt= no
default_md= sha256
distinguished_name = dn
[dn]
C = CA
ST = QC
L = Montreal
O = End Point
OU = Testing Domain
emailAddress = yourname@yourdomain.com
CN = localhost
```

- Using a text editor, copy paste the following into v3.ext

```
authorityKeyIdentifier = keyid, issuer
basicConstraints = CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost
```

Next we will generate your SSL certificates. Run these commands respectively in the terminal

- Generate a private key
 - Cmd: “*openssl genrsa -des3 -out ~/ssl/rootCA.key 2048*”

- After running this command you will be prompted to create a password

```
Edwins-MBP:ssl edwinyachoui$ openssl genrsa -des3 -out ~/ssl/rootCA.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for /Users/edwinyachoui/ssl/rootCA.key:
Verifying - Enter pass phrase for /Users/edwinyachoui/ssl/rootCA.key:
```

- Create root certificate

- Cmd: “`openssl req -x509 -new -nodes -key ~/ssl/rootCA.key -sha256 -days 1024 -out ~/ssl/rootCA.pem`”

```
Edwins-MBP:ssl edwinyachoui$ openssl req -x509 -new -nodes -key ~/ssl/rootCA.key -sha256 -days 1024 -out ~/ssl/rootCA.pem
Enter pass phrase for /Users/edwinyachoui/ssl/rootCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:CA
State or Province Name (full name) []:CA
Locality Name (eg, city) []:Ottawa
Organization Name (eg, company) []:Paul Point
Organizational Unit Name (eg, section) []:Teaching Division
Common Name (eg, fully qualified host name) []:localhost
Email Address []:edwin.yachoui@gmail.com
```

- Create the private key for the certificate (server.key)

- Cmd: “`openssl req -new -sha256 -nodes -out server.csr -newkey rsa:2048 -keyout server.key -config <(cat server.csr.cnf)`”

```
Edwins-MBP:ssl edwinyachoui$ openssl req -new -sha256 -nodes -out server.csr -newkey rsa:2048 -keyout server.key -config <(cat server.csr.cnf)
Generating a 2048 bit RSA private key
.....
writing new private key to 'server.key'
```

- Generate server.crt

- Cmd: “`openssl x509 -req -in server.csr -CA ~/ssl/rootCA.pem -CAkey ~/ssl/rootCA.key -CAcreateserial -out server.crt -days 500 -sha256 -extfile v3.ext`”

```
Edwins-MBP:ssl edwinyachoui$ openssl x509 -req -in server.csr -CA ~/ssl/rootCA.pem -CAkey ~/ssl/rootCA.key -CAcreateserial -out server.crt -days 500 -sha256 -extfile v3.ext
Certificate request self-signature ok
subject=C=CA,ST=CA,L=Ottawa,O=Paul Point,OU=Teaching Division,CN=localhost
Certificate CA Private Key
```

- Verify that everything is working properly (Notice the x509v3 Authority Key Identifier and DNS: localhost)

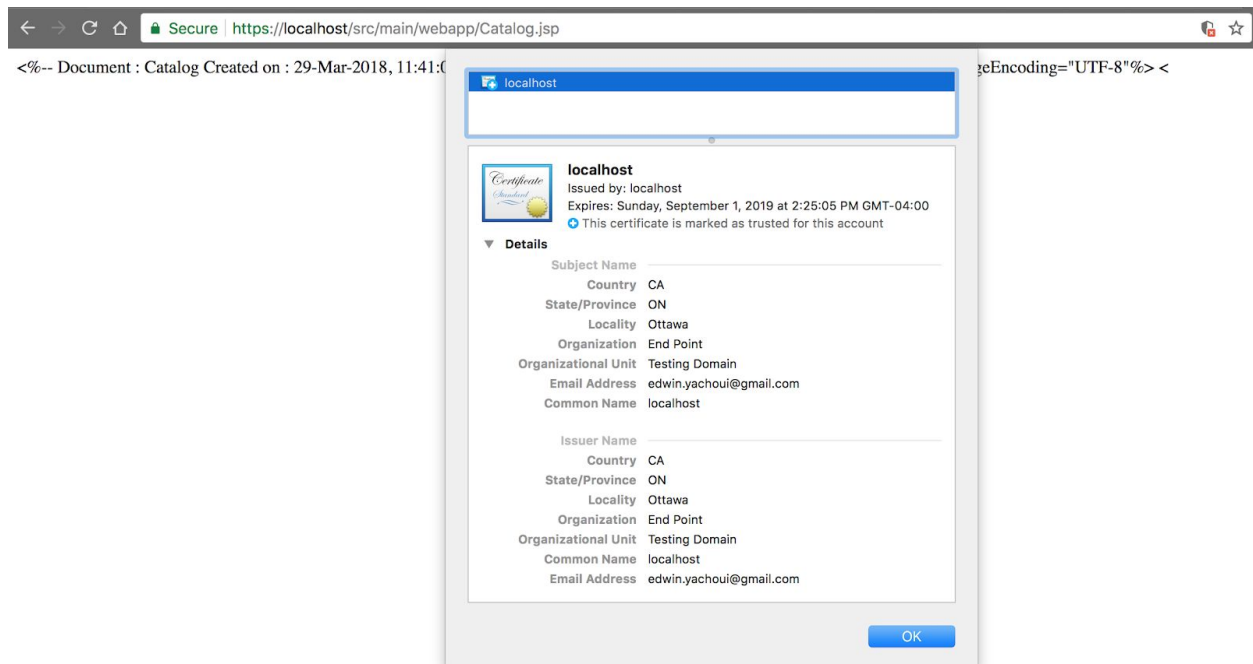
- [illegible]

- 45

- Change the path of SSLCertificateFile navigate to the “ssl” folder created earlier (“/Users/YourUsername/ssl/server.key”)
- Change the path of SSLCertificateKeyFile navigate to the ssl folder created earlier (“/Users/YourUsername/ssl/server.key”)
- Save all changes
- Restart MAMP Servers
- <https://localhost/MAMP/?language=English>
- Note: First time you load the page you should click on trust certificate.



- Example of Supply Chain application running over HTTPS (locally)



Logging Service

- Logs are stored based on User activity.
- Users can query logs by entering respective date and time

LOG

Something just happened? Check it out!

From

mm/dd/yyyy --:-- --

To

mm/dd/yyyy --:-- --

Search

Reset

Class Reference	Level	Message	Timestamp
soen487.service.WarehouseService	INFO	Start shipping items to customer.	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)
soen487.service.WarehouseService	INFO	Item #2 is available	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)
soen487.service.WarehouseService	INFO	Item #4 is available	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)
soen487.service.WarehouseService	INFO	Item #9 is available	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)
soen487.service.WarehouseService	INFO	Shipment size: 3	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)
soen487.service.WarehouseService	INFO	Shipment size: 3	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)
soen487.service.WarehouseService	INFO	Shipment size: 3	Fri Apr 20 2018 16:22:49 GMT-0400 (Eastern Daylight Time)

Performance Evaluation

Evaluation Choice

SOAP method shipGoods() from warehouseService and REST resource getCatalog() from RetailerService were chosen for the performance evaluation. The reason is that getCatalog() is invoked every time a user visits the site and whenever a user orders something shipGoods() would first to be invoked. Therefore, these two methods go hand in hand and we estimate would be a good place to find out where are the bottlenecks.

Test Parameters

We tested the SOAP method by sending a sample payload (in XML) to the warehouseService URI by POST to test the shipGood() invocation. If the server returns status 200 and response time < 200ms, then we consider it a success.

Similarly, we tested the REST method by calling the RESTful retailer API to evaluate the getCatalog() method. The successful response would consist of status 200, response time < 200ms, and a json response from the server containing the inventory status.

Test Cases

We tested the methods in three variation:

1. REST resource only
2. SOAP method only
3. REST resource with SOAP method together

For each test case, we performed 10 iterations and repeated the test 10 times, so in total of 100 tests.

Test Results and Analysis

REST resource only

If we were only invoking the RESTful API then all tests passed and the average response time was 10.72 ms.

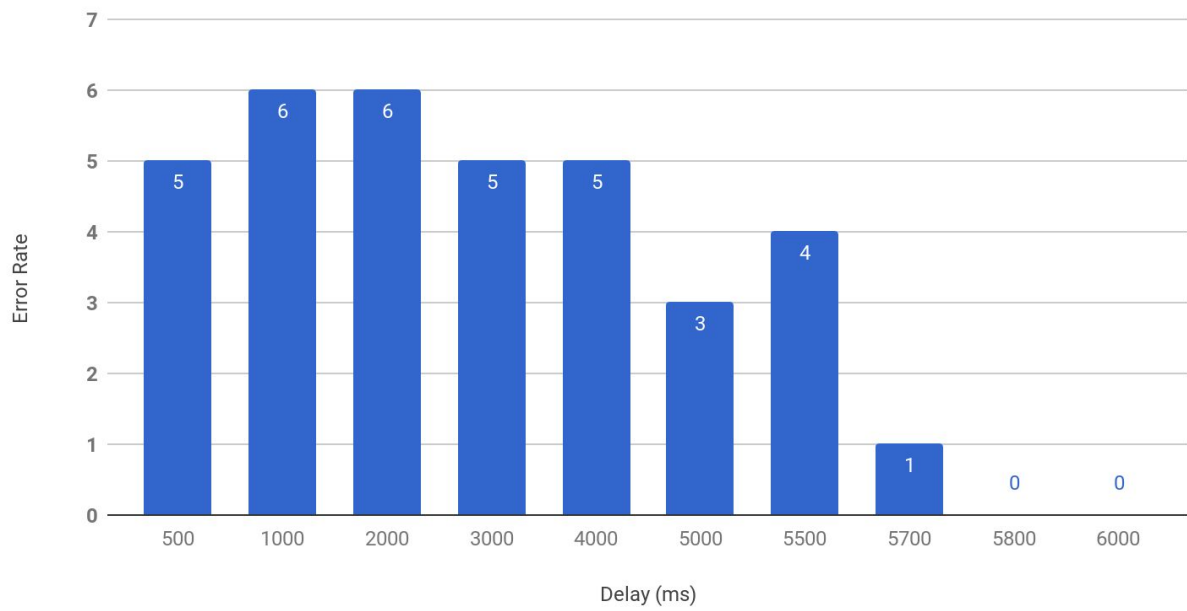
Test Type	Iterations	Delay (ms)	Response Time Average (ms)	Success	Fails
REST - HTTP	10	0	6.2	10	0
REST - HTTP	10	0	6.4	10	0
REST - HTTP	10	0	11.9	10	0
REST - HTTP	10	0	12	10	0
REST - HTTP	10	0	11.5	10	0
REST - HTTP	10	0	11.9	10	0
REST - HTTP	10	0	10.7	10	0
REST - HTTP	10	0	12.9	10	0

REST - HTTP	10	0	10.9	10	0
REST - HTTP	10	0	12.8	10	0
		Total Average:	10.72		

SOAP method only

We encountered bottlenecks when testing for the SOAP method. The Glassfish server couldn't keep up if the delay between invocations is < 5800ms. Over 50% of requests would failed if the invocation delay is \leq 4000ms. We believe the bottleneck is caused by the interactions of checking the warehouses, replenish the inventory with different warehouses etc. We estimate the problem will cascade even further if we were to simulate multiple items purchase.

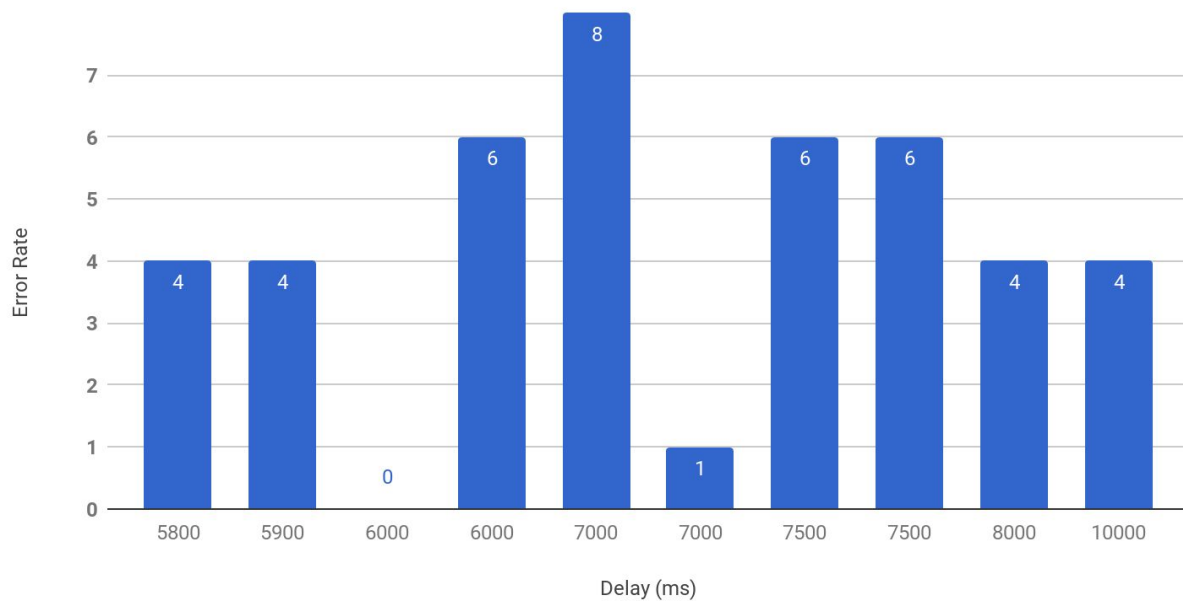
Test Type	Iterations	Delay (ms)	Response Time Average (ms)	Success	Fails
SOAP - HTTP	10	500	87.4	5	5
SOAP - HTTP	10	1000	146.5	4	6
SOAP - HTTP	10	2000	114.3	4	6
SOAP - HTTP	10	3000	124.8	5	5
SOAP - HTTP	10	4000	125.8	5	5
SOAP - HTTP	10	5000	118.6	7	3
SOAP - HTTP	10	5500	141.2	6	4
SOAP - HTTP	10	5700	128.1	9	1
SOAP - HTTP	10	5800	140.6	10	0
SOAP - HTTP	10	6000	113.4	10	0
		Total Average:	124.07		



REST resource with SOAP method together

Since there is a bottleneck in SOAP methods, we also observed similar bottlenecks when testing SOAP and RESTful API together. However, making matters worse, the long process time with the SOAP methods is affecting the RESTful API making it unstable. We observed that even when the RESTful API was functioning the response time were often in the range of 1500ms. In addition, we did not observe the correlation between increasing the delay between the invocation and lower fail rate (mainly the RESTful API) unlike the test case with SOAP method only.

Test Type	Iterations	Delay (ms)	Response Time Average (ms)	Success	Fails
REST & SOAP - HTTP	10	5800	612.8	16	4
REST & SOAP - HTTP	10	5900	582.4	16	4
REST & SOAP - HTTP	10	6000	815.0	20	0
REST & SOAP - HTTP	10	6000	494.7	14	6
REST & SOAP - HTTP	10	7000	279.9	12	8
REST & SOAP - HTTP	10	7000	814.0	19	1
REST & SOAP - HTTP	10	7500	463.8	14	6
REST & SOAP - HTTP	10	7500	459.2	14	6
REST & SOAP - HTTP	10	8000	614.3	16	4
REST & SOAP - HTTP	10	10000	599.1	16	4
		Total Average:	573.5		



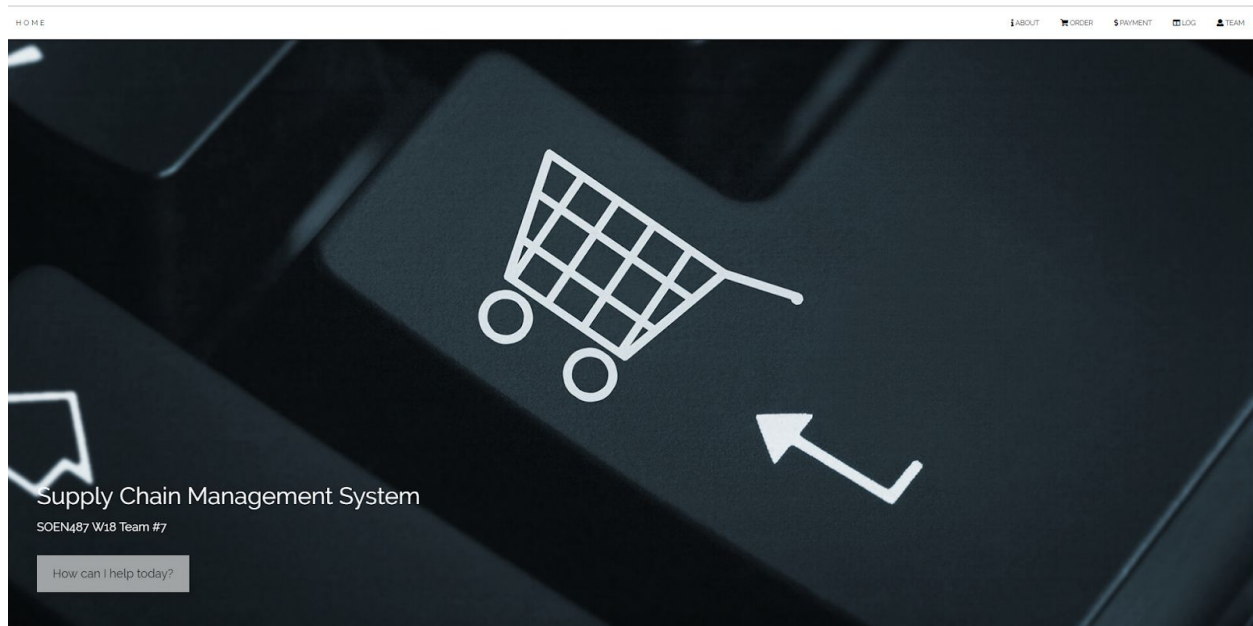
Test Type	Test # - Iterations #	Delay (ms)	Response Time (ms)	Success?	Fails?
REST API	2 - 1	6000	32	Y	N
SOAP Method	2 - 1	6000	147	Y	N
REST API	2 - 2	6000	1650	Y	N
SOAP Method	2 - 2	6000	73	Y	N
REST API	2 - 3	6000	1766	Y	N
SOAP Method	2 - 3	6000	94	Y	N
REST API	2 - 4	6000	1725	Y	N
SOAP Method	2 - 4	6000	82	Y	N
REST API	2 - 5	6000	1583	Y	N
SOAP Method	2 - 5	6000	72	Y	N
REST API	2 - 6	6000	1581	Y	N
SOAP Method	2 - 6	6000	70	Y	N
REST API	2 - 7	6000	1633	Y	N
SOAP Method	2 - 7	6000	65	Y	N
REST API	2 - 8	6000	1989	Y	N
SOAP Method	2 - 8	6000	67	Y	N
REST API	2 - 9	6000	1776	Y	N

SOAP Method	2 - 9	6000	73	Y	N
REST API	2 - 10	6000	1754	Y	N
SOAP Method	2 - 10	6000	64	Y	N

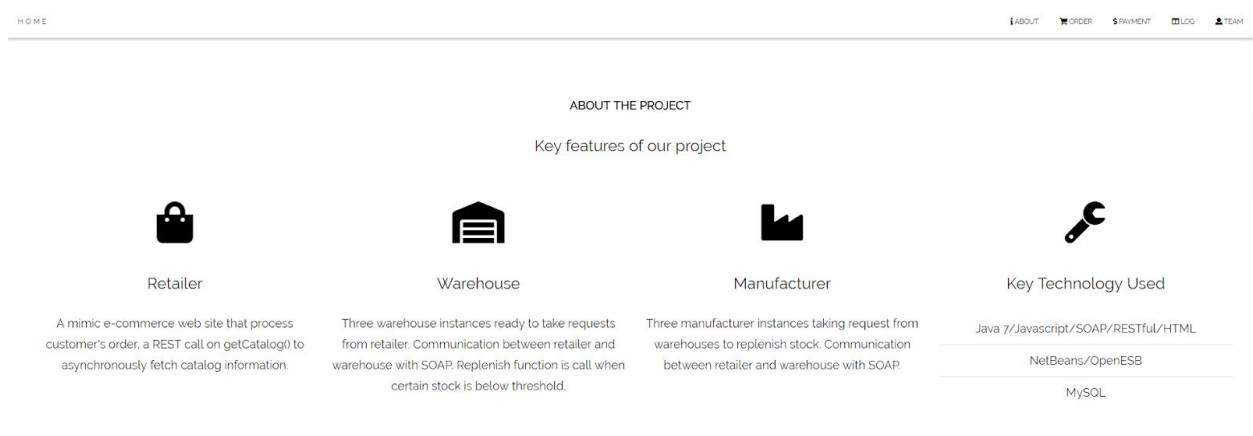
Test Type	Test # - Iterations #	Delay (ms)	Response Time (ms)	Success?	Fails?
REST API	3 - 1	6000	41	Y	N
SOAP Method	3 - 1	6000	356	Y	N
REST API	3 - 2	6000	9	N	Y
SOAP Method	3 - 2	6000	155	Y	N
REST API	3 - 3	6000	4	N	Y
SOAP Method	3 - 3	6000	145	Y	N
REST API	3 - 4	6000	1757	Y	N
SOAP Method	3 - 4	6000	82	Y	N
REST API	3 - 5	6000	5	N	Y
SOAP Method	3 - 5	6000	133	Y	N
REST API	3 - 6	6000	6	N	Y
SOAP Method	3 - 6	6000	144	Y	N
REST API	3 - 7	6000	1730	Y	N
SOAP Method	3 - 7	6000	136	Y	N
REST API	3 - 8	6000	875	N	Y
SOAP Method	3 - 8	6000	101	Y	N
REST API	3 - 9	6000	89	N	Y
SOAP Method	3 - 9	6000	94	Y	N
REST API	3 - 10	6000	1976	Y	N
SOAP Method	3 - 10	6000	76	Y	N

SC Web Application

Home



About



Order

HOME

ABOUTORDER\$ PAYMENTLOGTEAM

Order

Feel like buying?

Product ID	Warehouse ID	Manufacturer	Product Type	Unit Price	Quantity
1	1	Sony	TV	659.98	197
2	1	Panasonic	Video_Camera	139.99	177
3	1	Samsung	DVD_Player	39.99	95
4	2	Sony	TV	659.98	200
5	2	Panasonic	Video_Camera	139.99	70
6	2	Samsung	DVD_Player	39.99	128
7	3	Sony	TV	659.98	165
8	3	Panasonic	Video_Camera	139.99	200
9	3	Samsung	DVD_Player	39.99	54

Purchase

Video_Camera

DVD_Player

TV

55

20

10

Name

Mohammed Raza

Street 1

23 rue vinci

Street 2

City

Montreal

State

Quebec

Zip

H3B 2H2

Country

Canada

Submit

Payment

HOME

ABOUTORDER\$ PAYMENTLOGTEAM

PAYMENT

Thanks you for your business.

Customer Reference	Amount Due	Paid in full	Last Update
13835	1979.94	false	Fri Apr 20 2018 18:44:11 GMT-0400 (Eastern Daylight Time)
27449	0	true	Fri Apr 20 2018 17:56:14 GMT-0400 (Eastern Daylight Time)
49597	15099.05	false	Fri Apr 20 2018 19:00:01 GMT-0400 (Eastern Daylight Time)

SC Pay

Customer Reference

13835

Payment


1979.94

Pay

Team

THE TEAM


The ones who rocks



Yang Shen

Team Lead


Student ID 2769330



Mohammed Raza

Team member


Student ID 2646092



Edwin Khalil Yachoui

Team member


Student ID 2133442



Marco Dagostino

Team member

Student ID 29687357



Tsang Chi Kit

Team member

Student ID 25692636

56

Individual Contribution

	Mohammed Raza	Yang Shen	Marco Dagostino	Edwin Khalil Yachoui	Tsang Chi Kit
PM1	- Warehouse SOAP Service and Servlet/JSP Client - Integration	-Manufacturer SOAP and corresponding servlet/client -Integration	1.2	1.1	- Warehouse SOAP Service and Servlet/JSP Client Integration
PM2	- RESTful API for Warehouse Service - Warehouse Java Client Integration	- RESTful API Java Client Integration	2.1	-Retailer Service -Retailer Client	- 2.2 - RESTful API for Manufacturer Service
PM3	-3.1 -3.3.SC AI -Integration	Integration and Documentation	3.3 Documentation	-Retailer Catalog	- Retailer Service and Servlet/JSP Integration
PM4	-Docker -Mashup -Integration -Documentation	-Logging Service -Re-construct entire project. -Dead code cleanup -Integration - Documentation	-4.1 -Documentation	-Security HTTPS -Documentation	- Design for Scalability: Hadoop/ MapReduce - Create Test Suite and Performance Evaluation - Documentation

References

[1] Project Instructions, Appendix A, Serguei A. Mokhov.

[2] Lecture Notes 18 by Osama Abu-Obeid