Faculty of Engineeeing and Technology

Master in Software Engineering Program

Service-Oriented Software Engineering

Course project report
Service Oriented Architecture and implementation

for Inventory Management System and Nutrition Client

Prepared by
Rami Khalyleh
Mohammad Shawahneh
Noura Houshieh

Instructor
Dr. Nariman Ammar

Contents

ABSTRACT 3

Keywords 3

INTRODUCTION 3

IMS DESCRIPTION 6

Web Services 7

Motivation 7

DESIGN 8

- Architecture 9
- Paradigm 10

Technologies 11

Implementation 11

- Overview 11
- Elements of IMS 13
- Swagger APIs documentation 17
- Class diagram for the model 20

Deployment 21

Testing 22

Non-functional requirements 22

THE CLIENT FOR THE OTHER PROJECT 22

- Architecture 23
- Technologies 23
- Service composition 24
  - Service orchestration 24
  - Choreography 25
- Client Application location 26

Discussion 26

Conclusion 27

Appendices 28

- Appendix A 28

Appendix B 31

ABSTRACT

The inventory management system (IMS) is structured as a service oriented system, it is exposed as web services, each subsystem of the IMS is structured and exposed as a standalone service that can be consumed by any client or other service independently, The services are controlled by an orchestration service controlling the workflow and logic, the orchestration of the services is responsible of managing the web services composition. Also, to integrate the services from outside server. The services are exposed as Restful services, represented in JSON data, the architectural details below specifies the system high level and overall structure and the communication of services.

Keywords

Spring; API; Composition; Inventory system

INTRODUCTION

Web service is a standardized way that used to integrated web applications over internet protocol using XML, WSDL, REST, and UDDL. Using web services let user to access available software over the internet. XML is provide a protocol to communicate between different technologies of software and tags data. It offer application-to-application interaction between programs, object, message or document. Client send a request contains XML messaging to Server and wait for response. These communications are not strict to any operating systems or programming languages. For instance, client that use windows OS can talk with others whose use Unix OS or who use Java languages can talk with who use C languages. In web services, application can be local, distributed, or web-based also implement web standards such as TCP/IP, HTTP, Java, and HTML

Web services depends on SOAP and REST standards to transfer a message when a client ask for services. SOAP ia protocol while REST is architectural style. In SOAP, there is a tightly coupled to the serve. Its implement using a contract between client and server. This Contract can be loss of any part change its status. Whereas, REST seems as browser. It is less coupling. Any modification on a service API through adding new methods did not violate the standard. Client needs to know how to use service using SOAP but in REST client can be use the service without previous knowledge. The main strong point that you can implement REST using SOAP because REST is a concept but vice versa is wrong because SOAP is a protocol.

In order to expose business logic, SOAP depends on service interface but REST use URL. Web services using SOAP builds in JAX-WS is the java and needs more bandwidth and resource. It also, define its security and use only XML standard format. Therefore, it is less preferred from a community. By contract, REST depends on JAX-RS and needs less bandwidth and resources. It   inherits security measures from the underlying transport. In addition, use various data format to represent data in text, HTML, XML and JSON. Therefore, it’s more preferred from a community.

RESTFUL standards is use when you use CRUD-targeted in simple implementation. Main RESTful problem is that it's CRUD-targeted, therefore it would be not the best choice for complex logic implementation and then you can use REST alone.

REST is a set of architectural guidelines expressed as Resource-Oriented Architecture (ROA). SOA is an approach used for building distributed systems. These systems deliver its functionality to client as a service or for other services. Service oriented Architecture stack divide in two element, see Figure.1. These elements are functional and the second is quality of service

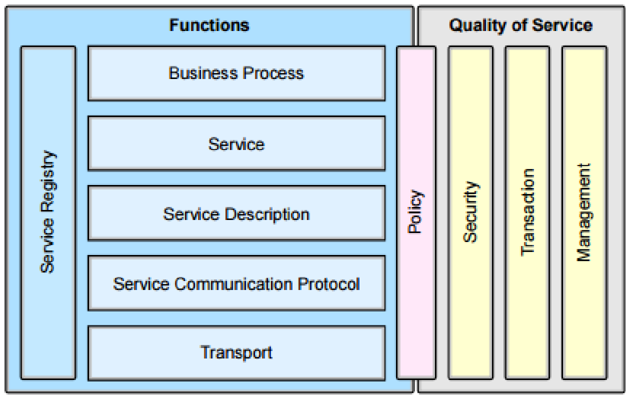


Figure 1:Element of Service Oriented Architecture

The functional part include six aspects. First is Transport which is a techniques used to move the service order from sender to receiver and vice verse. Second is service Communication protocol which is a techniques used for provide communication between sender and receiver. Third is service description which is used to provide a scheme of documentation about the service and the way can be used to invoke also how to invoke the data that needed to use the service. Fourth is service which is describe an actual service that can the client use it.

Fifth is business process it's a service act as a collection of service and the sequences between them that let to make a multiple process composed with other service in different application. Sixth is a service registry which is a repository of the services and its description that let a service provider to publish it and the consumer to discover it.

However, the quality service part includes four aspects. First is a policy, which is a set of rules that make the service provider available to consumers. Second is security, which is a set of rules that used for identification, authentication and access control. Third is transaction, which is a set of attributes that might be applied of service to gain consistent result. Fourth is Management, which is a set of attribute that used to manage services.

The collaboration in SOA find, bind and invoke . a client   query the registry to find the service. If the registry find the service, it invoke the service according to the information in the service description.. Then, it will create an interface between client and provider in order to bind a service. Each service has interface to allow it to be published, discovered and invoked. In addition, each service can composed from different services

Using SOA provide a set of benefits, see Figure 2. It provide layer of abstraction that let to leverage existing assets as a service without rebuilt them from scratch. SOA allow easy integration, manage the system complexities through providing a specification about the services, and build the assets in discrete system. Moreover, Composition a new service in different of existing make a faster responsive. In addition, it reduce the cost and increase reuse. The most advantage benefit is SOA allow business be ready in future.

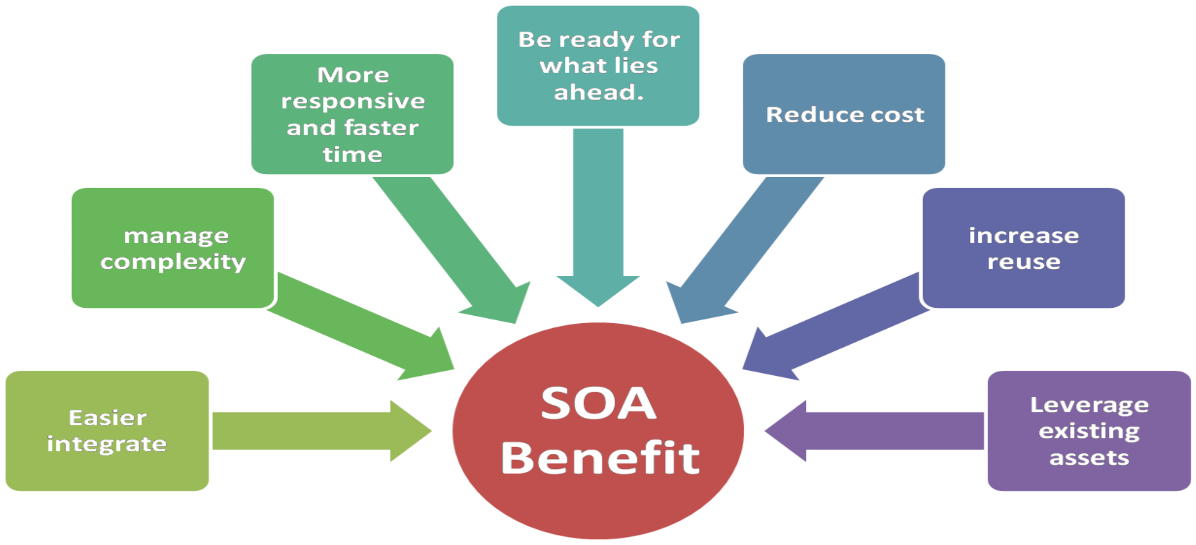


Figure 2: SOA Benefits

This Report is intended to provide the reader an overview of the project concepts, subsystems, services, and high-level service oriented architecture of the system as well as motivation of using such architecture. Moreover, This Report introduces high level of details as well as services and components description, the API specifications will be a subject of later document. In addition, we implement an inventory management system (IMS system using SOA. IMS exposed as RESTFul services and represented in JSON data. The rest of report show IMS description, architecture and our implementation to IMS services using SOA.

IMS DESCRIPTION

The inventory management system (IMS) is responsible for managing full inventory system, it is exposed to the user as web and mobile system, IMS is structured as subsystem, each of which is responsible for managing and controlling part of the system, such that the user might consume one or more service for executing one functionality, independent operations are executed in parallel, whereas dependent services are executed sequentially, orchestration service is composing other services according to the predefined workflows achieving the desired logic. IMS is expected to manage the warehouse, orders, customers, suppliers, shipment, mail, GPS, mail system, and catalog system. The services will expose its data independently and orchestrated in workflows. The subsystems of IMS are:

- Payment subsystem, it includes invoices, payment methods, calculating order payment quantity, and defining payment validity.
- Warehouse subsystem, this subsystem is responsible of managing and tracking warehouse items, as well as exposing the items for the authenticated client
- Shipment subsystem, which is responsible about shipping the package to the customer.
- GPS subsystem, which is responsible for tracking the shipped packages.
- Mail subsystem, which is used to communicate with customers and departments.
- Ordering subsystem, which is used to manage orders.
- Personnel subsystem, which is responsible for managing the personnel who is executing the tasks assigned.
- Authentication subsystem, which has a responsibility to authenticate operation.
- Customers’ subsystem, this system holds information about customers of the system.
- Product Catalog subsystem, this system holds detailed information about items, products, categories, history of items, suppliers as detailed below:
    - Configure product types and classify them into subtypes such as assets, components, and consumables.
    - Categorize all products.

- Create a record of all products in environment with necessary details such as price, tax rates, warranty, and depreciation rates.
- Associate every product with a vendor and a maintenance vendor for easy tracking.

Web Services

A web service is an autonomous entity executing specific functionality and exposing it over web technologies, usually HTTP, from other perspective a web service is a piece of software or a software module responsible of doing a desired computation over web, communicating with it happens with a messaging system, by any kind of representation like XML, JSON or any other representation.

There are two common types of web services, namely simple object access protocol (SOAP) and representational state transfer (REST). The web services can be exposed by wide range of software development technologies, Java and C sharp are widely used to implement and provide such services, exposing those services over cloud systems enables the scalability, maintainability, extensibility and public visibility of those services, beside performance and throughput of the services.

WSDL and WADL are used to describe web services, for REST services, documentation and specification are provided by other means like swagger which is used in our project.

Web services are modular, self-contained, distributed and dynamic, enable the client to access them over internet or network, and enable many client types like mobile application, web and desktop applications.

Web services are a collection of open protocols and standards used for exchanging data among heterogeneous systems, such that they are solving architectural mismatches and break the coupling with technologies, during our project client and server many technologies were used which totally different, Java, frontend web languages, C sharp, cloud services, Google cloud and app engine, benefiting from the independency nature of web services and service oriented architecture.

Service composition is another concept which refers to consuming one service from another service or client, during our project we have applied this concept on client side by composing services to provide desired client behavior , and in backend IMS calling internal services and web SOAP services.

Motivation

The system which we are describing and architecting in this report is well known traditional system, which is used and implemented in many technologies and architectures, the main contribution here is having it as a service oriented system, we picked a known application domain so as to keep our focus on the SOA part of the system. However, the motivations of having IMS as service-oriented system over traditional system can be summarized in the following:

- Modularity: the proposed system supports high level of modularity and abstraction through using of independent services and orchestration.
- Ownership: this architecture provides well-known and well-defined ownership of each service. Multiple teams can work independently on the same project.
- Decoupling between the producer services and the service consumers, and to the underlying service implementation and logic.

- Maintainability and extensibility, which are highly supported by this architecture as we it is providing a high-level separation of concerns beside defined flows and responsibilities.
- Robustness and Fault Isolation: each service has its own module of exception and fault handling, which enables the system to react in more rebus way for faults and errors, like retrying the failed part only or waiting for services recovery.
- High Scalability: the proposed design supports scalability such that any service can be replicated over any required number of cloud nodes.
- Efficiency of development life cycle: having the system in separated independent services supports having efficient development life cycle as each team member can pick one service and finish it then all the work will be integrated easily in the orchestration and work flows. However, there is an expected complexity in handling operations and workflows but that can be resolved by means of having well defined documentation for each service.

DESIGN

The proposed design of ISM is based on SOA, it is a decomposition of the system into atomic independent web services, the services integrate and communicate with each other by the orchestration service, which is responsible of controlling the workflows, it is like an engine who has all the power and information to run other services in order to execute the desired requested functionality, it's responsible for combining and invoking the services, below is the high level architectural diagram, and elements description.

Architecture

We used service-oriented architecture to implement ISM system. See Figure.3, which shows architecture. In ISM architecture, each subsystem provide as distinct service also each of them have different interface and description. Using SOA, the system provide composition and integration between these subsystem in between or with other outsides system. Swagger was implemented to provide orchestration between these compost services. The client request a composition service sequences using RESTFUL API and JSON data type. In addition, receive the response in JSON. In addition, Customer service access the service on the demands in standard way using method invocations.

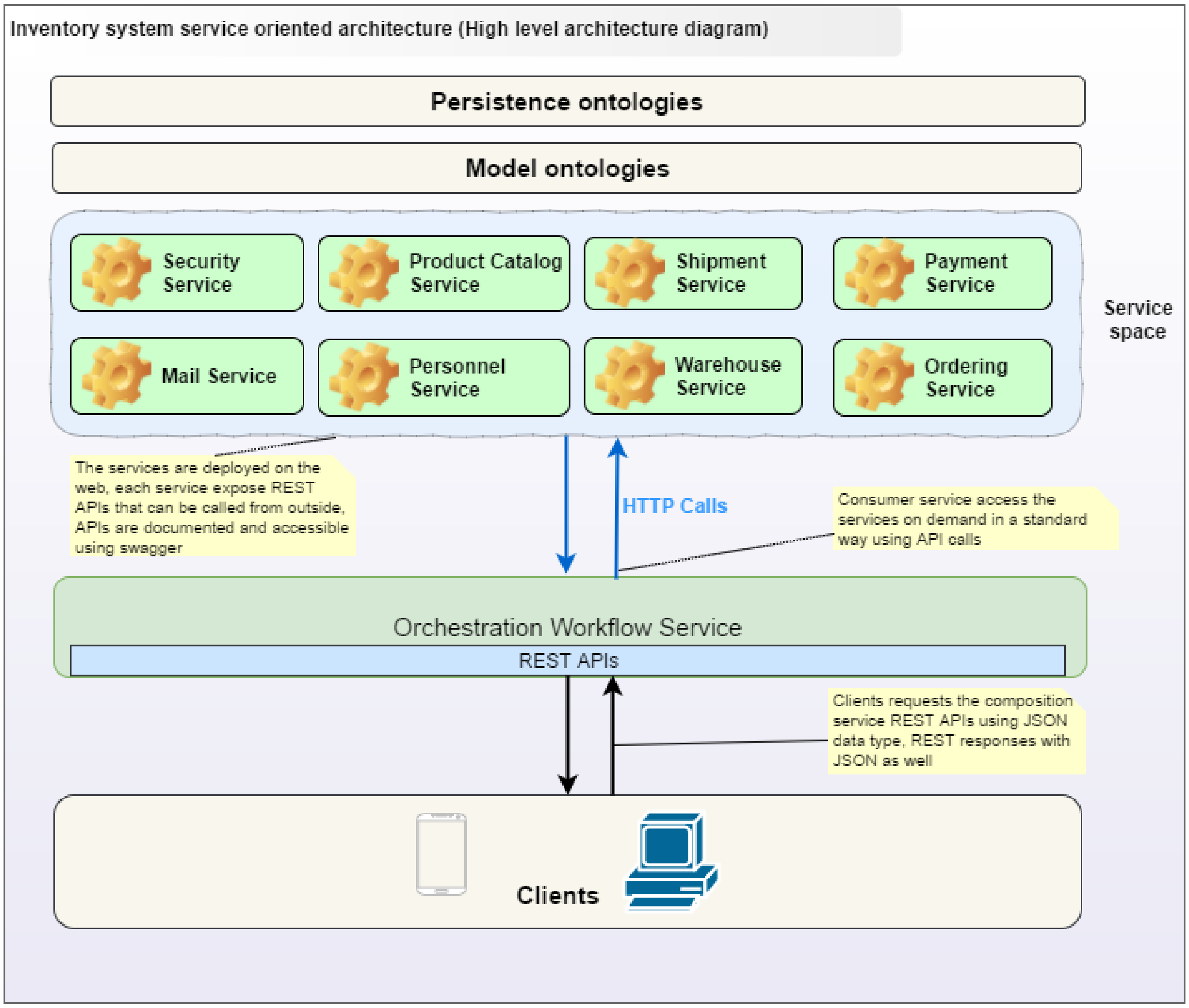

Inventory system service oriented architecture (High level architecture diagram)
Persistence ontologies
Model ontologies
Security Service
Product Catalog Service
Shipment Service
Payment Service
Mail Service
Personnel Service
Warehouse Service
Ordering Service
Service space
The services are deployed on the web, each service expose REST APIs that can be called from outside, APIs are documented and accessible using swagger
HTTP Calls
Consumer service access the services on demand in a standard way using API calls
Orchestration Workflow Service
REST APIs
Clients requests the composition service REST APIs using JSON data type, REST responses with JSON as well
Clients

Figure 3: High-level Architecture Diagram

In Figure .4, shows one of the workflow as a sample, which has been executed in the orchestration service, the client order an item. The workflow orchestration show how the orchestration combining and invoking the needed service to perform the order item request.

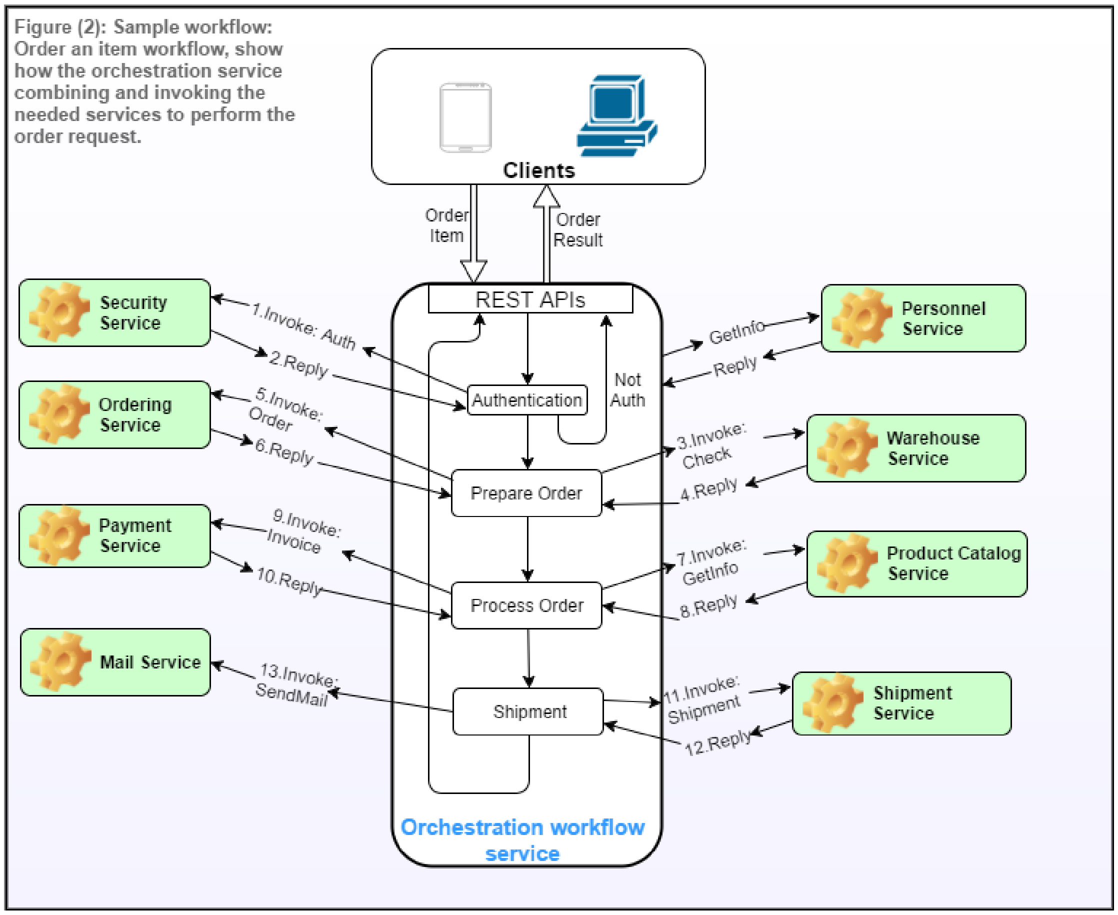


Figure 4: Orchestration service

Paradigm

The IMS services will be exposed as REST services, most of them will be RESTFul , but that will not be a constraint as there would be a level of flexibility during implementation. At this point of design, we could not find any need to use SOAP, but during implementation, we might need some to consume some external SOAP services, we will keep the design document updated.

Technologies

Numerous technologies are used in our IMS project, and client online nutrition services app, which was developed by our team, here, is a list of technologies used, short detentions, benefits, and rationality of the decision.

- Java: which is used as a main technology for developing the IMS web services, it is selected due to its powerful APIs, team strong experience in it, maturity of REST API implementation and exposing in java, community associations, tools availability, free nature of it and of most supportive tools and frameworks, beside many other language capabilities.
- Spring boot: Which is a brand new framework derived from Spring, and made development process easier and faster, as it is originally designed to simplify the bootstrapping of spring applications, and as it is introducing an opinionated approach for application configurations, our projects have used it in order to benefit from those capabilities, which helped in building a rapid development and integration process.
- Data Store APIs: in order to provide our prototype an in memory data store service was developed based on java capabilities, which enables us to concentrate on the core of the application and the web services side, decreasing the complexities of managing storage APIs.
- Cloud services: a cloud service is any service available to the users on demand over internet via a cloud service provider, cloud services are used to support scalability, maintainability and public exposing of the APIs, we have exposed our inventory services as cloud, such that our clients have used the APIs without any coupling to our development process life cycle, also we have developed the client over cloud benefiting from the power of cloud to integrate with the Nutrition services, and simplifying the deployment and demonstration of the client application.
- SOAP client: it is a spring boot mechanism which enables consuming SOAP services easily and quickly, we have used this technology in order to composite the currency convertor SOAP services from web which is specified in the implementation section.
- Swagger: is a specification technology for documenting REST APIs, it gives full specification about implemented REST APIs, such as URI, method, representation and any other details developer needs to expose to the APIs consumer, swagger also enables API testing mechanism easily and quickly, swagger user interface provides a web interface. During our project swagger user interface is exposed to the IMS APIs clients making their integration with our APIs efficient and effective.
- Web frontend technologies: the client application is using JQuery as a main front end framework, beside, HTML, JavaScript, CSS and Ajax technologies, all of them have collaborated to produce a robust and reliable client web application which consumes the Nutrition services.

Implementation

Overview

The IMS is implemented as a spring boot model view controller MVC application as specified:

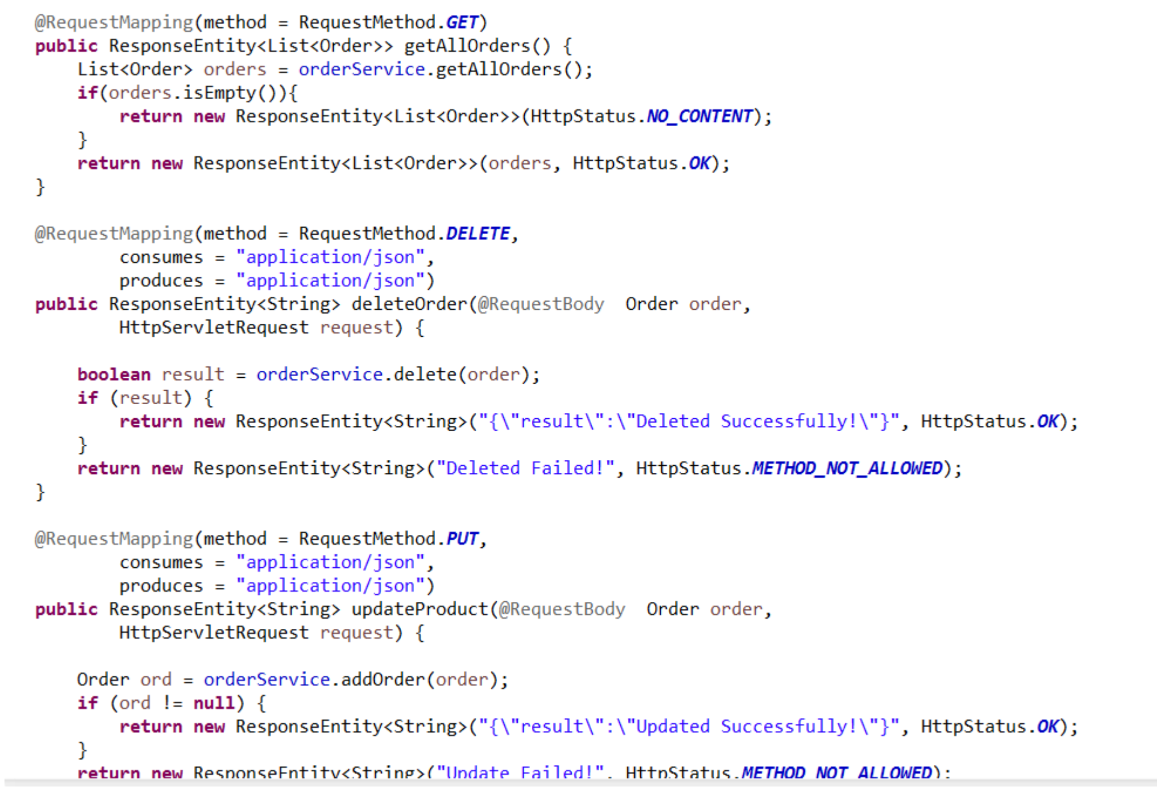
View/ representation: which is the web services consumer, which is Postman during the development of the application and the client application which is developed by our client team who will demonstrate the application user interface and use cases, the representation of the web services are all in JSON format which makes it easier for web frontend development.

Controller: the controller represent the REST APIs exposed, such that the IMS has used spring boot annotations @RestController in order to provide the desired APIs in the desired HTTP method, the methods included GET, POST, PUT and delete as specified in the swagger API section, the APIs consumed JSON from the web services consumer, and produces JSON as well as the representation of the web services response, the client passed request is validated against the APIs internal logic, processed and responds in the desired manner comprising the interfaces for the application server, and the web services implementation.

The Spring technology supports the implementation of REST controllers by list of annotations which we have used extensively to produce a reliable application, below is a figure 5 shows one controller, the order controller, which exposes the order system REST APIs.

The @RequestMapping annotation is the entry point for declaring a rest method, by which the web service controller can expose its desired REST API, specifying the representation and any other details.

The internal implementation of the controller depends on the logic layer which is the IMS internal services layer, which is responsible of managing the logic and providing the data to the controller which in turn passes the data to the API consumer.



```java
@RequestMapping(method = RequestMethod.GET)
public ResponseEntity<List<Order>> getAllOrders() {
    List<Order> orders = orderService.getAllOrders();
    if(orders.isEmpty()){
        return new ResponseEntity<List<Order>>(HttpStatus.NO_CONTENT);
    }
    return new ResponseEntity<List<Order>>(orders, HttpStatus.OK);
}

@RequestMapping(method = RequestMethod.DELETE,
        consumes = "application/json",
        produces = "application/json")
public ResponseEntity<String> deleteOrder(@RequestBody  Order order,
        HttpServletRequest request) {

    boolean result = orderService.delete(order);
    if (result) {
        return new ResponseEntity<String>("{\"result\":\"Deleted Successfully!\"}", HttpStatus.OK);
    }
    return new ResponseEntity<String>("Deleted Failed!", HttpStatus.METHOD_NOT_ALLOWED);
}

@RequestMapping(method = RequestMethod.PUT,
        consumes = "application/json",
        produces = "application/json")
public ResponseEntity<String> updateProduct(@RequestBody  Order order,
        HttpServletRequest request) {

    Order ord = orderService.addOrder(order);
    if (ord != null) {
        return new ResponseEntity<String>("{\"result\":\"Updated Successfully!\"}", HttpStatus.OK);
    }
    return new ResponseEntity<String>("Update Failed!", HttpStatus.METHOD_NOT_ALLOWED);
```

Figure 5 REST controller sample

Model: the model layer consists of list of entities and model classes that collaborate together to comprise the application behavior when processed in the related services, which are in turn comprise the controller of the models, the models elements is specified in the model section.

In Spring boot application the model plays a crucial rule in simplifying the application and REST APIs system, such that the REST APIs can be viewed as models communicating with the client or consumer application, which makes more controller for the web services layer, and for the other low level services managing and processing the data.

Elements of IMS

The elements of the IMS system can be categorized as controllers, data models, autonomous services, composed services, and orchestration, below are the details of those elements:

Controllers:

All controllers below are used for exposing REST APIs, using the above mentioned spring technology annotations and other facilities provided by the spring boot framework.

- Catalog controller: it is an interface and API for supplier sub system details and other related issues to the supplying of the inventory systems.
- Customer controller: an interface and controller responsible for customer creation, update of information, retrieval and deletions.
- Employee controller: which manages the employees of the inventory systems, by creation , retrieval update and deletions of employees of IMS system.
- Product controller: it is responsible of managing product, product is used to categorize the items, such that more than one item can refer to one product, this controller helps API consumer to create, update, list or delete products.
- Item controller: It is responsibility is concentrated in managing the system items, such that the API consumer can list existing items and use them for his order, create new item and add it to the stock, update its information quantity, or price, and delete it.
- Payment controller: this controller enables consumer to define the payment details, method and process, get list of available payments, update the details or delete an existing payment.
- Shipment: in order to deliver an order a shipment should be defined, this controller enables this process by enabling creation of shipment, or listing existing payments as well as deleting them.
- Order controller: this controller comprises the orchestration consumer controller as it is dependent of many other controllers causing the usage of many other system services, by using the call methodology from the order service.

Services:

- Currency SOAP service**:** in order to calculate the total amount of the order, a currency convertor SOAP service is used, this service is doing the Spring boot internal technology of consuming web services in order to provide the functionality of consuming the SOAP service, this functionality is executed by adding the WSDL in the pom file, then Spring boot framework generates the code according to WSDL specification enabled us to use the generated classes in order to consume the service and expose the conversion rate to the order orchestration services by means of method invocation and procedure calls.
- Order service: is the logical computation of order system, it is a kind of orchestration and service composition as it is a consumer of the currency SOAP service and of other services in the system like product and warehouse services.
- Orchestration service: which is responsible of calling the Nutrition services to collect information about employees and their health information.
- Catalog service: is responsible about logic processing of the catalogs.
- Item service: is responsible about managing operations and storage of item logic.

- Product service: is responsible about product manipulations.
- Customer service: is responsible about customer information, manipulations and persistence.
- Payment service: handles the payment logic, persistence and processing.
- Shipment service: handles shipment information and data.
- Persistence service: represents the DAO and in memory data store system of the whole application, which is composed in other services.

Services deployment and source code:

The services are deployed online, in the following locations (With swagger UI):

- All services hosted on:
  http://31.168.143.199:8080/manager/html

- `Items service:`
  `https://github.com/rkhalyleh/items-service.git`
  `http://31.168.143.199:8080/itemsService/swagger-ui.html`

- `Payments service:`
  `https://github.com/rkhalyleh/payments-service.git`
  `http://31.168.143.199:8080/paymentsService/swagger-ui.html`

- `Shipment service:`
  `https://github.com/rkhalyleh/shipment-service.git`
  `http://31.168.143.199:8080/shipmentService/swagger-ui.html`

- `Warehouse service:`
  `https://github.com/rkhalyleh/warehouse-service.git`
  `http://31.168.143.199:8080/warehouseService/swagger-ui.html`

Data models

The system has many data model elements, here are the basic elements, and during the implementation, other data models will be provided:

- Customer: basic information about customer, name, address, contact information, it is used in the communication with the client and inside application data communications.
- Payment: amount, method and other characteristics of the payment, it is a core part of having complete order.
- Item: name, quantity, sell price, buying price, place in warehouse, it is the core of the system, which is composed in the order orchestration system.
- Order: id, customer, items, and other details, order is the main intent of the system which is the main scenario that other models try to achieve it and support it.
- Product: name, price, tax, warranty, depreciation rate, supplier, the product is a category of the items.
- Supplier: name, basic information, address, types supplied, it is used to track the suppliers of the inventory system.

- Employee: the basic model for personnel service, by which the shipment and the order can be carried out.
- Shipment: the shipment staff for the order, it is used to complete the order and composed as part of the order sub system and services.

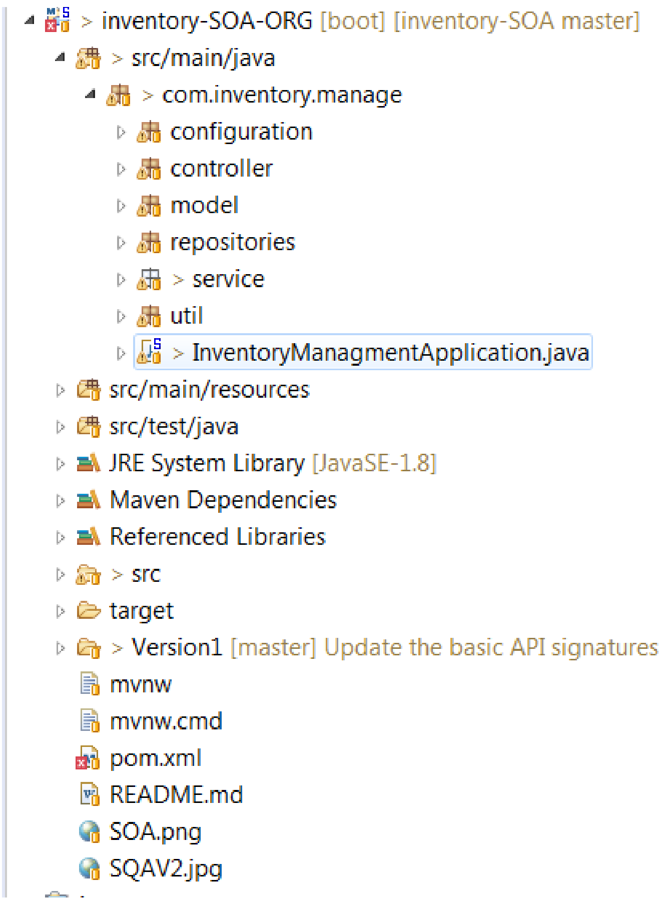
For each type of data model , the basic operation was offered in ISM system are Create , read, update and delete (CRUD). Also, the data representation was used to represent each model is JSON, automatically by Spring technology.

Relations

The relations among data models and services are achieved by direct composition inside models and by calls and invocation of services in order to provide the desired behavior.

Project packages

Here is figure 7 a screenshot of the main packaging of the web services IMS app, other screenshot are document in appendix B.

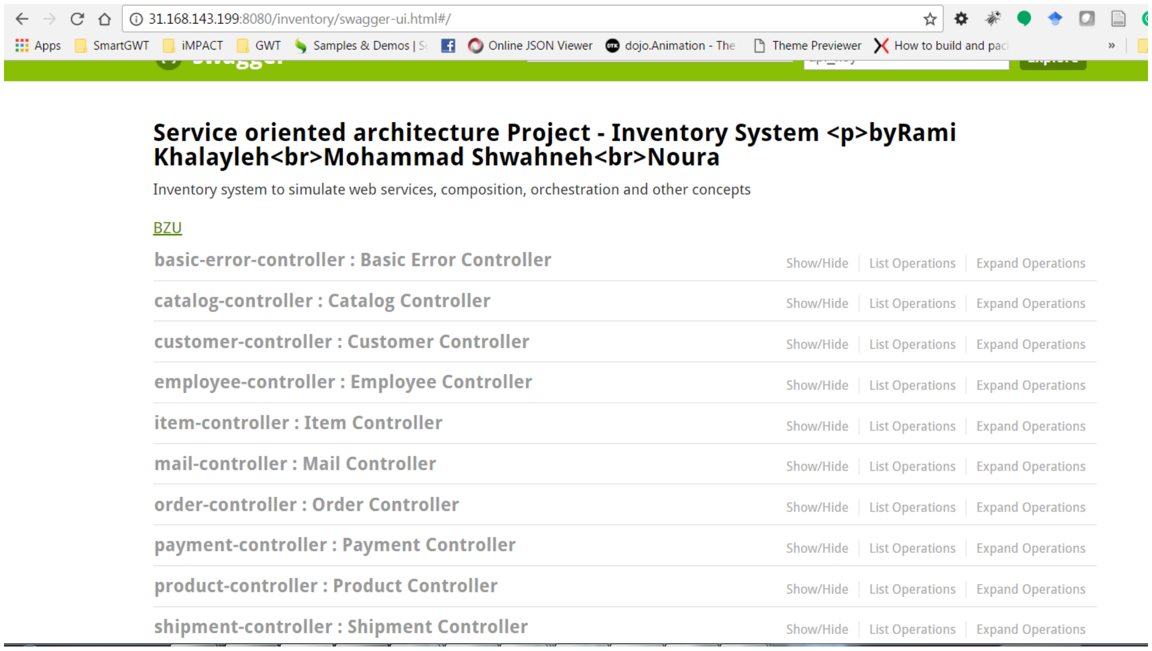


```
> inventory-SOA-ORG [boot] [inventory-SOA master]
  > src/main/java
    > com.inventory.manage
      configuration
      controller
      model
      repositories
      > service
      util
      > InventoryManagmentApplication.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  Referenced Libraries
  > src
  target
  > Version1 [master] Update the basic API signatures
  mvnw
  mvnw.cmd
  pom.xml
  README.md
  SOA.png
  SQAV2.jpg
```

Figure 7 web services main application packaging

Swagger APIs documentation

Swagger is implemented to show all the APIs of the system and to enable the client developers to integrate easily with the web services implemented, here are few screens from swagger of our deployed web services:

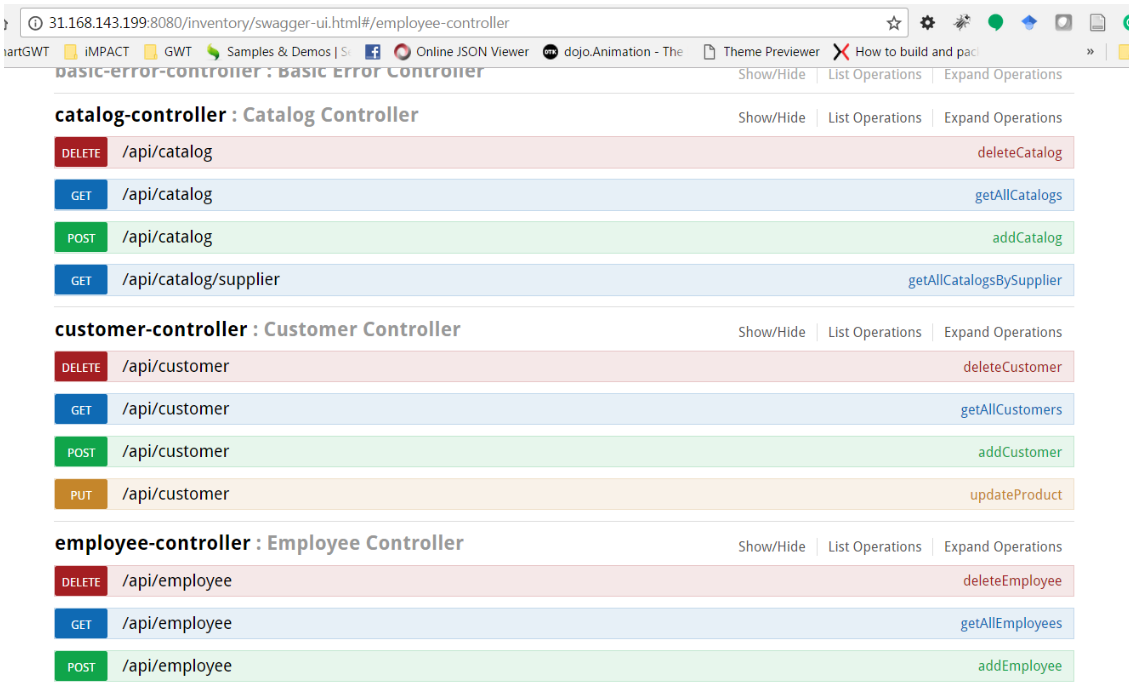


**Service oriented architecture Project - Inventory System <p>byRami Khalayleh
Mohammad Shwahneh
Noura**

Inventory system to simulate web services, composition, orchestration and other concepts

BZU

basic-error-controller : Basic Error Controller	Show/Hide | List Operations | Expand Operations
catalog-controller : Catalog Controller	Show/Hide | List Operations | Expand Operations
customer-controller : Customer Controller	Show/Hide | List Operations | Expand Operations
employee-controller : Employee Controller	Show/Hide | List Operations | Expand Operations
item-controller : Item Controller	Show/Hide | List Operations | Expand Operations
mail-controller : Mail Controller	Show/Hide | List Operations | Expand Operations
order-controller : Order Controller	Show/Hide | List Operations | Expand Operations
payment-controller : Payment Controller	Show/Hide | List Operations | Expand Operations
product-controller : Product Controller	Show/Hide | List Operations | Expand Operations
shipment-controller : Shipment Controller	Show/Hide | List Operations | Expand Operations



basic-error-controller : Basic Error Controller — Show/Hide | List Operations | Expand Operations

catalog-controller : Catalog Controller — Show/Hide | List Operations | Expand Operations

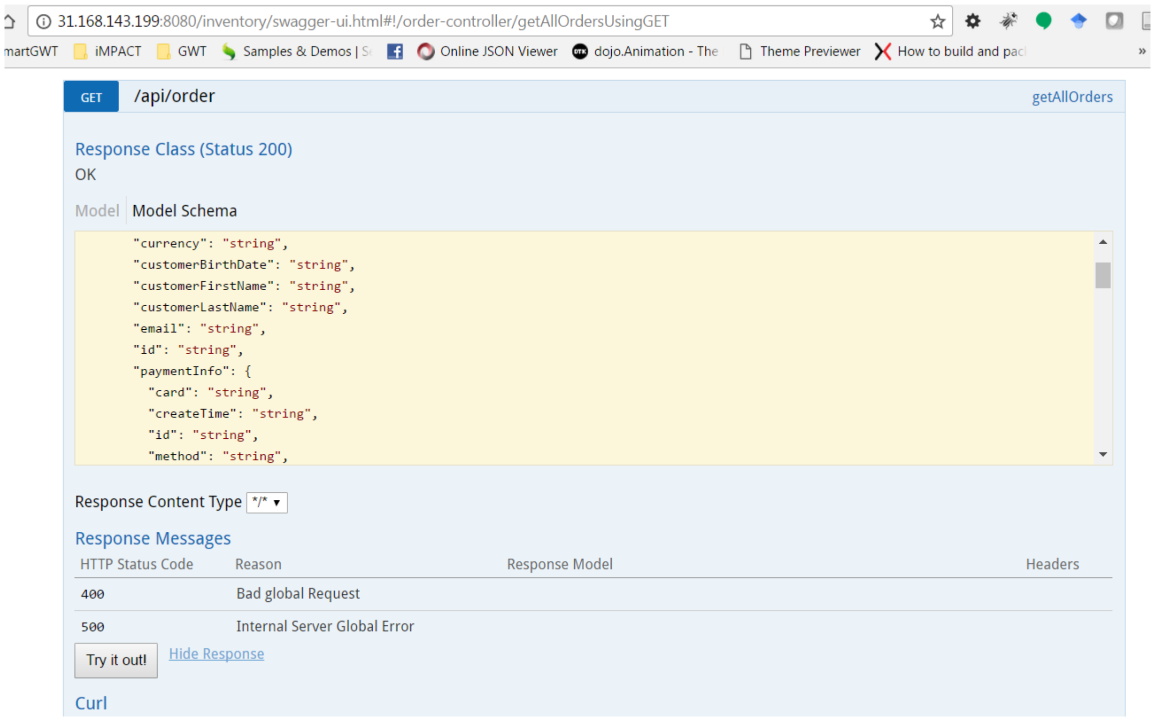
DELETE	/api/catalog	deleteCatalog
GET	/api/catalog	getAllCatalogs
POST	/api/catalog	addCatalog
GET	/api/catalog/supplier	getAllCatalogsBySupplier

customer-controller : Customer Controller — Show/Hide | List Operations | Expand Operations

DELETE	/api/customer	deleteCustomer
GET	/api/customer	getAllCustomers
POST	/api/customer	addCustomer
PUT	/api/customer	updateProduct

employee-controller : Employee Controller — Show/Hide | List Operations | Expand Operations

DELETE	/api/employee	deleteEmployee
GET	/api/employee	getAllEmployees
POST	/api/employee	addEmployee



31.168.143.199:8080/inventory/swagger-ui.html#!/order-controller/getAllOrdersUsingGET

GET /api/order getAllOrders

Response Class (Status 200)

OK

Model | Model Schema

```
        "currency": "string",
        "customerBirthDate": "string",
        "customerFirstName": "string",
        "customerLastName": "string",
        "email": "string",
        "id": "string",
        "paymentInfo": {
          "card": "string",
          "createTime": "string",
          "id": "string",
          "method": "string",
```

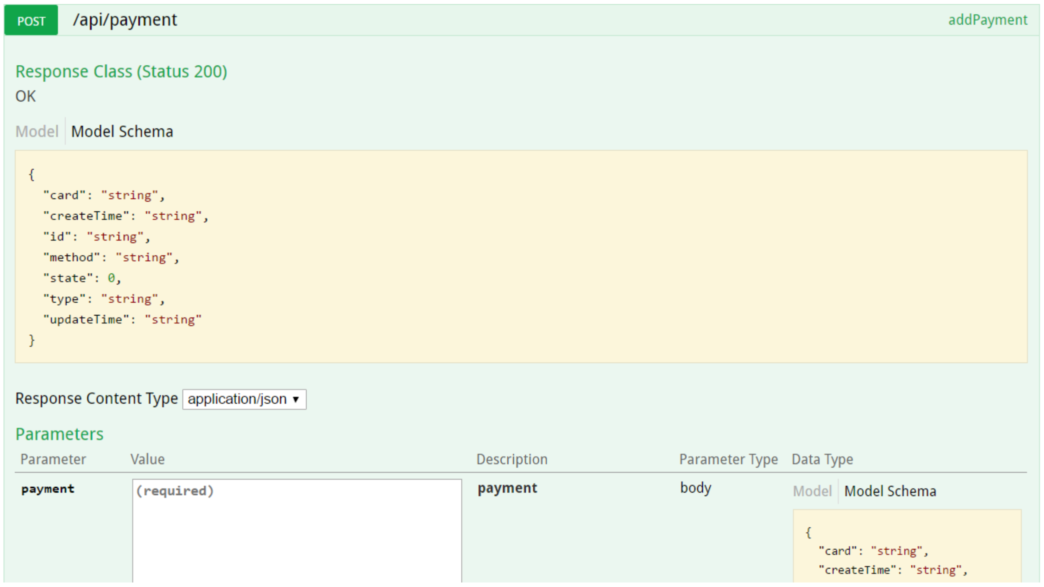
Response Content Type */*

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	Bad global Request		
500	Internal Server Global Error		

Try it out! Hide Response

Curl



POST /api/payment addPayment

Response Class (Status 200)
OK

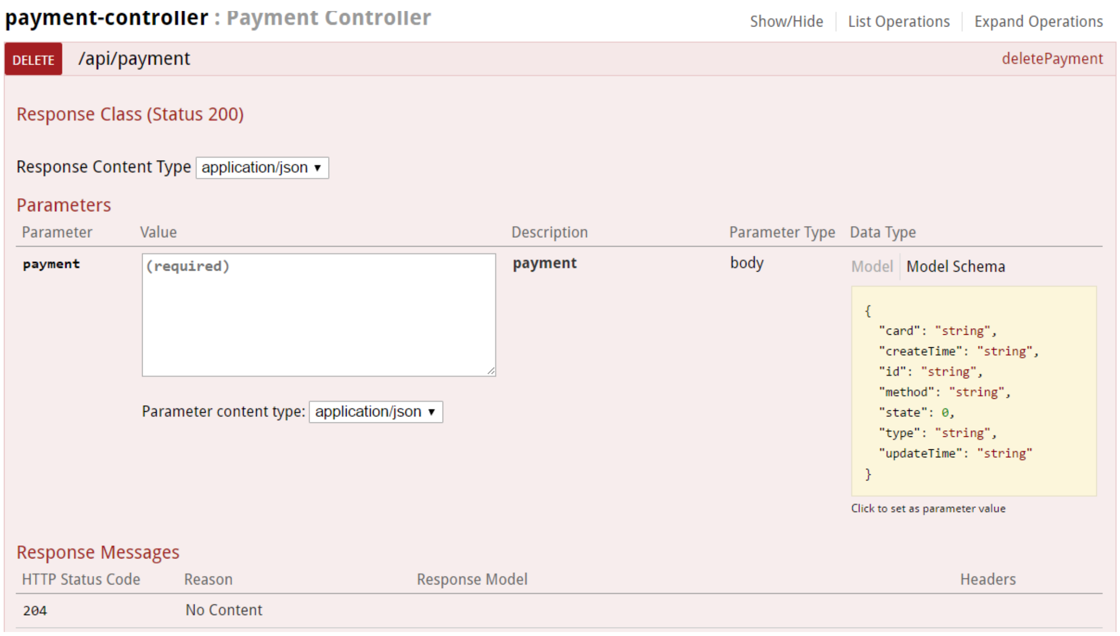
Model | Model Schema

```
{
  "card": "string",
  "createTime": "string",
  "id": "string",
  "method": "string",
  "state": 0,
  "type": "string",
  "updateTime": "string"
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
payment	(required)	payment	body	Model Model Schema `{ "card": "string", "createTime": "string",`



payment-controller : Payment Controller Show/Hide | List Operations | Expand Operations

DELETE /api/payment deletePayment

Response Class (Status 200)

Response Content Type application/json

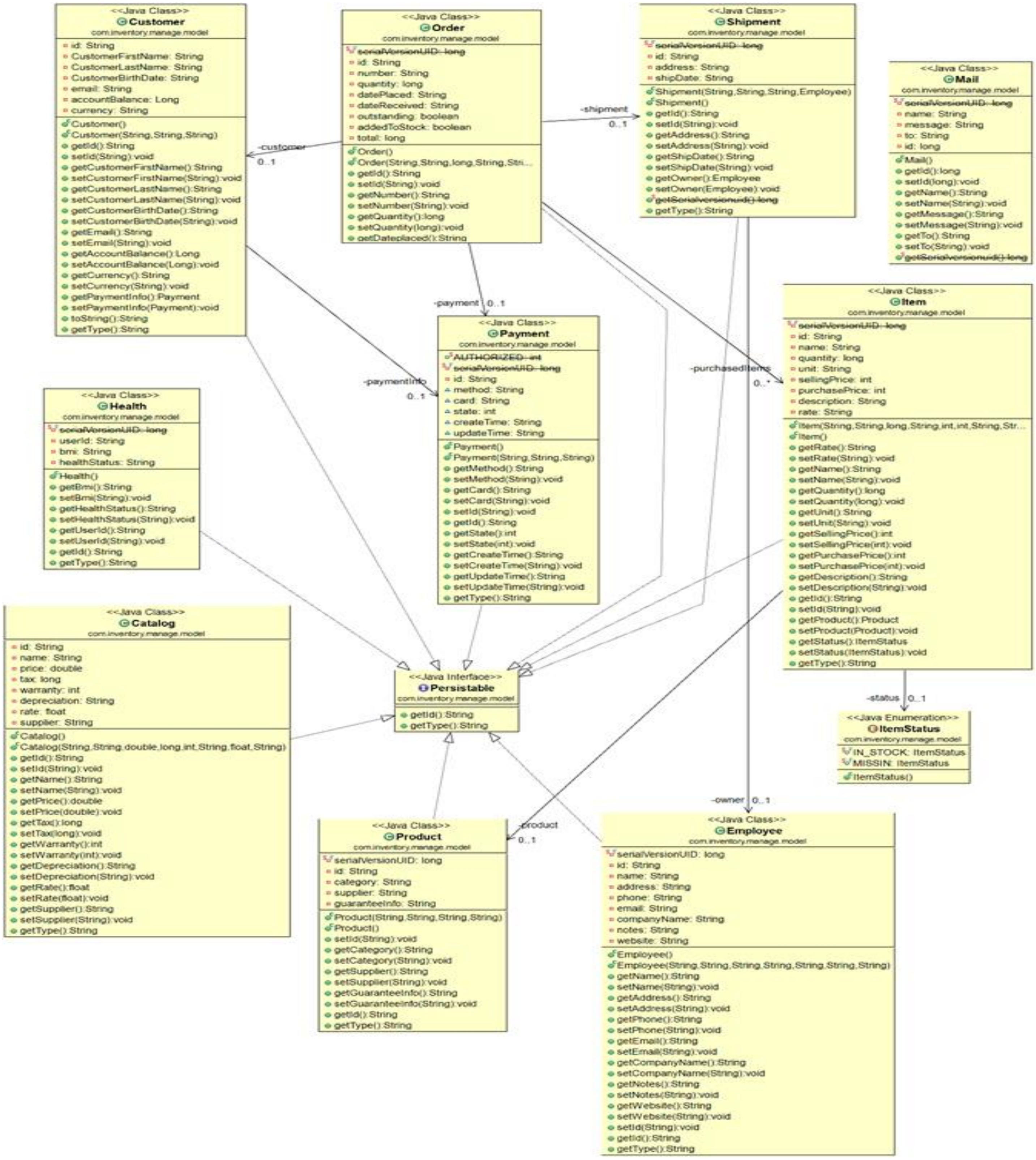
Parameters

Parameter	Value	Description	Parameter Type	Data Type
payment	(required) Parameter content type: application/json	payment	body	Model Model Schema `{ "card": "string", "createTime": "string", "id": "string", "method": "string", "state": 0, "type": "string", "updateTime": "string" }` Click to set as parameter value

Response Messages

HTTP Status Code	Reason	Response Model	Headers
204	No Content		

Class diagram for the model



Deployment

The IMS app is deployed over cloud services, using VPS system, the application is packaged as a war, enabling easy packaging using maven technology and easy deployment using web tool of tomcat server and Linux file service.

The application is hosted on GitHub, which supports team collaboration and source code management, as well as easy deployment, during development. The cloud instance was provided by a local ISP, then the VPS instance was prepared with Java 8 and tomcat, having it ready to use in no time.

Having the application as Spring boot maven project enabled fast deployment to the cloud services, using maven spring plugins directly during the development time, and using the war web packaging during the production time, that made the process and the deployment more reliable and robust, as well as supporting of the maintainability of the application.

The IMS web services are deployed to application is deployed to:

http://31.168.143.199:8080/inventory/

The swagger API specification is deployed on:

http://31.168.143.199:8080/inventory/swagger-ui.html

Figure 6 below shows the deployment view:

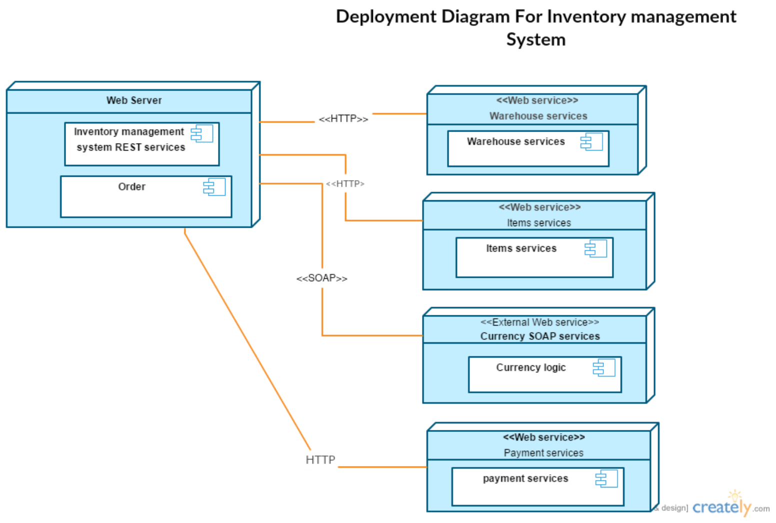


Figure 6 Deployment diagram

Testing

The IMS application and web services were testing accumulatively during the application development and at the end of the project. Testing depended mainly on native browser calls for REST GET APIs, which provide quick and easy way.

Postman was used as well to test all kinds of implemented REST APIs, GET/PUT/POST and DELETE, enabling debugging and hot update of issues appeared during development.

Swagger HOT testing of API is used as well in order to validate the APIs requests and responses as well, benefiting from swagger features.

Non-functional requirements

IMS design and implementation have targeted a list of non-functional requirements in order to provide usable application and web services, those requirements included reliability, maintainability, usability, reusability, interoperability, performance and other quality attributes.

The performance and reliability metrics were monitored and tested at all levels of development causing to do some enhancements on the implementation during the development life cycle, and reaching a reliable robust web services at the end of the project, such that our client is totally satisfied and did not report any more issues, while our client has reported few issues at the beginning of his integration, we were able to fix reported issues and deploy them within few minutes which showed the power of maintainability and supportability of our IMS, exposed web services and implemented architecture.

THE CLIENT FOR THE OTHER PROJECT

We have implemented client application for the NutriServe project that exposes REST APIs to be used by the clients. We have created our application to be a consumer service to the NutriServe service provider. We have created a maven web project that divided into the following packages:

- Model: which contains the fundamental entities to interaction with the NutriServe project REST APIs (POST, PUT and GET data), and used to return these data entities to the UI client to be shown for the end users.
- Orchestration service: Contains the needed logic to connect to the NutriServe services to POST, PUT and GET data, and make it available for other modules in the application to use.
- Services: Contains the needed REST APIs that are exposed and used by the UI client module. REST APIs interact with the orchestration service module to get the data.
- UI: Contains the UI pages to present the data and functionality of the NutriServe services using the REST APIs implemented in the services module. The UI has been implemented using JQuery, AJAX, HTML and CSS technologies.

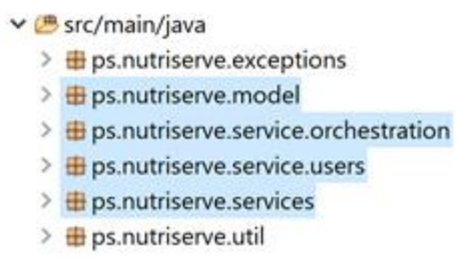
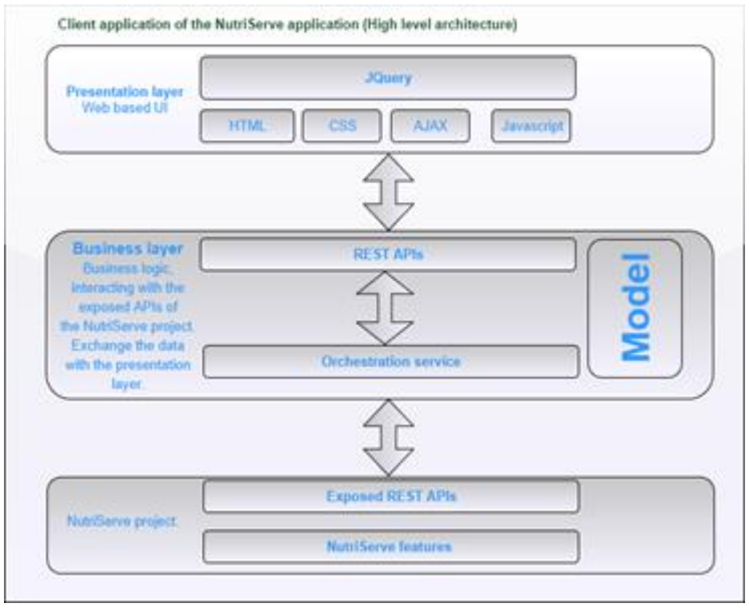

```
src/main/java
  ps.nutriserve.exceptions
  ps.nutriserve.model
  ps.nutriserve.service.orchestration
  ps.nutriserve.service.users
  ps.nutriserve.services
  ps.nutriserve.util
```

Figure (5): Client project main packages

Architecture

As a high level architecture, we have been used the n-tier architecture in implementing the application, it consists from three main layers, which are the presentation layer that is responsible to interact with the users, and business layer that is responsible to connect to the NutruServe exposed REST APIs.


Client application of the NutriServe application (High level architecture)
Presentation layer
Web based UI
JQuery
HTML
CSS
AJAX
Javascript
Business layer
Business logic.
Interacting with the exposed APIs of the NutriServe project.
Exchange the data with the presentation layer.
REST APIs
Orchestration service
Model
NutriServe project
Exposed REST APIs
NutriServe features

Technologies

We have used the following main technologies in implementing the project:

- Maven: We have created the project skeleton using maven, which is responsible structure and load the needed libraries in the project, build the project.
- Spring framework: we used the spring framework to implement the REST APIs and dependency injection in the application.

- JQuery: we used JQuery in implementing the UI pages that are presents the data and the functionality to the end users, it uses the exposed REST APIs from the application (point 2) through AJAX calls.

Service composition

In the client application, we have implemented a service composition to aggregate the inputs and data more than one web services and send it to the caller, which is the REST API to be rendered on UI views. On UI, we created a view to present this composite handles result. In the code, we have the class "ps.nutriserve.service.orchestration.HealthInfoImpl" which contains API "getUserHealthInfo" that calls three NutruServe services and handle the results then return it for the caller.

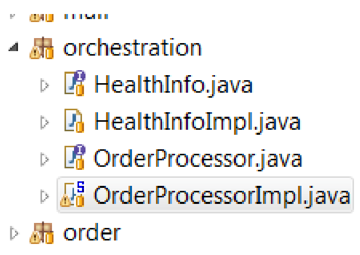
In inventory management system service composition is applied by orchestration and choreography methodologies as specified below.

Service orchestration

Service orchestration is applied in order system, during creation of an order, the order service needs to request many services from other web services over HTTP calls, since those services are deployed separately and should be part of the workflow using an orchestration mechanism as appeared in the conceptual architecture diagram, order orchestration component execute orchestration as flows:

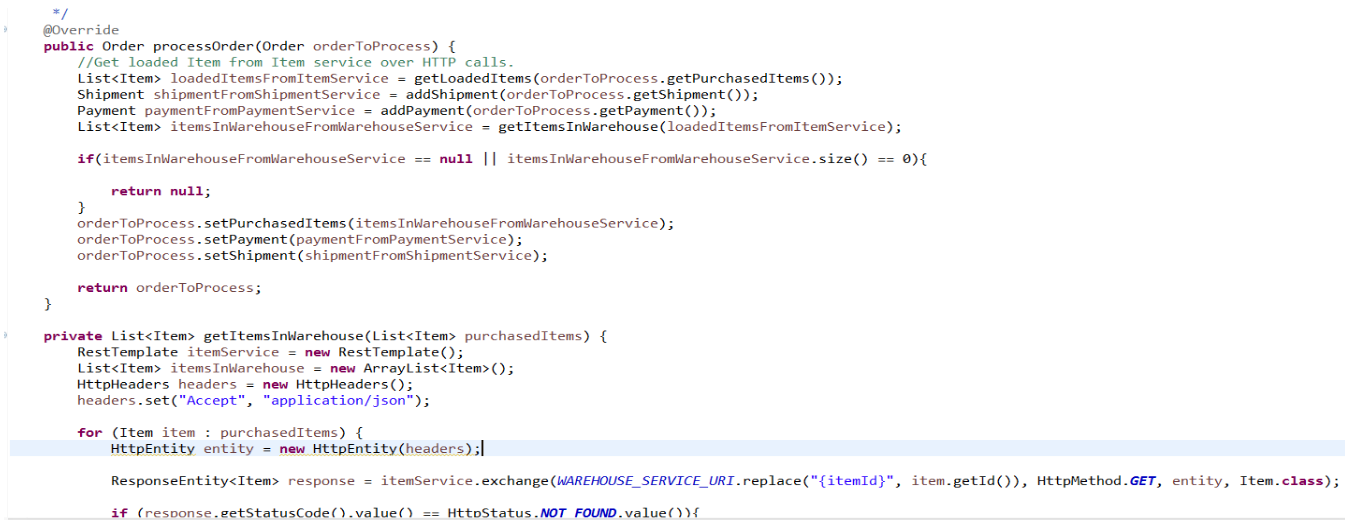
- Call Items service using HTTP to load available item details, any item not available will be dropped from the order.
- Call the shipment service using HTTP calls to add the requested shipment.
- Call the payment service over HTTP to add the claimed payment if valid.
- Update the order if all is valid and return it to the API layer to complete processing and communicating with the client.

The figure below shows the orchestration package of the inventory project:



Orchestration package figure.

Figure below shows code snippet of orchestration class:



```
     */
    @Override
    public Order processOrder(Order orderToProcess) {
        //Get loaded Item from Item service over HTTP calls.
        List<Item> loadedItemsFromItemService = getLoadedItems(orderToProcess.getPurchasedItems());
        Shipment shipmentFromShipmentService = addShipment(orderToProcess.getShipment());
        Payment paymentFromPaymentService = addPayment(orderToProcess.getPayment());
        List<Item> itemsInWarehouseFromWarehouseService = getItemsInWarehouse(loadedItemsFromItemService);

        if(itemsInWarehouseFromWarehouseService == null || itemsInWarehouseFromWarehouseService.size() == 0){

            return null;
        }
        orderToProcess.setPurchasedItems(itemsInWarehouseFromWarehouseService);
        orderToProcess.setPayment(paymentFromPaymentService);
        orderToProcess.setShipment(shipmentFromShipmentService);

        return orderToProcess;
    }

    private List<Item> getItemsInWarehouse(List<Item> purchasedItems) {
        RestTemplate itemService = new RestTemplate();
        List<Item> itemsInWarehouse = new ArrayList<Item>();
        HttpHeaders headers = new HttpHeaders();
        headers.set("Accept", "application/json");

        for (Item item : purchasedItems) {
            HttpEntity entity = new HttpEntity(headers);

            ResponseEntity<Item> response = itemService.exchange(WAREHOUSE_SERVICE_URI.replace("{itemId}", item.getId()), HttpMethod.GET, entity, Item.class);

            if (response.getStatusCode().value() == HttpStatus.NOT_FOUND.value()){
```

Order orchestration code snippet.

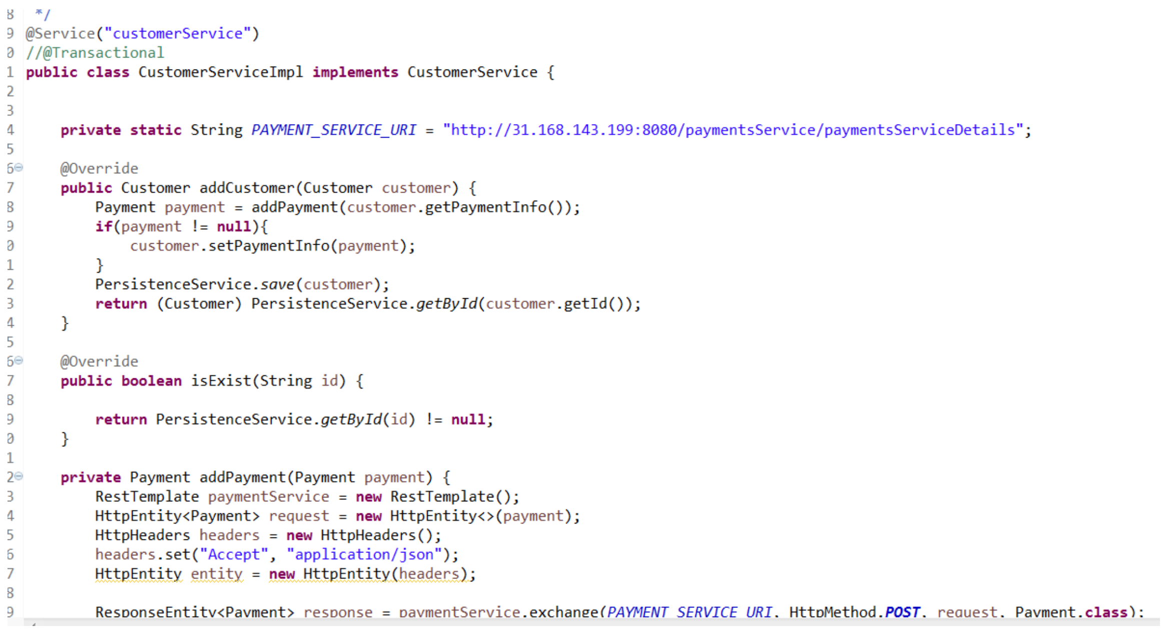
Using this orchestration mechanism, order service has applied the composition mechanism and provided enriched functionality depending on other independent services over HTTP calls.

Choreography

Choreography is a distributed integration logic that describes how web services need to interact with each other, at implementation level each service should know how to interact at message level with the services it needs.

In our inventory system customer service implemented choreography mechanisms by interacting directly with payment services over HTTP calls, in order to add the payment information for the requested customer.

Screenshot below shows the customer service implementation:



```java
 */
@Service("customerService")
//@Transactional
public class CustomerServiceImpl implements CustomerService {


    private static String PAYMENT_SERVICE_URI = "http://31.168.143.199:8080/paymentsService/paymentsServiceDetails";

    @Override
    public Customer addCustomer(Customer customer) {
        Payment payment = addPayment(customer.getPaymentInfo());
        if(payment != null){
            customer.setPaymentInfo(payment);
        }
        PersistenceService.save(customer);
        return (Customer) PersistenceService.getById(customer.getId());
    }

    @Override
    public boolean isExist(String id) {

        return PersistenceService.getById(id) != null;
    }

    private Payment addPayment(Payment payment) {
        RestTemplate paymentService = new RestTemplate();
        HttpEntity<Payment> request = new HttpEntity<>(payment);
        HttpHeaders headers = new HttpHeaders();
        headers.set("Accept", "application/json");
        HttpEntity entity = new HttpEntity(headers);

        ResponseEntity<Payment> response = paymentService.exchange(PAYMENT_SERVICE_URI, HttpMethod.POST, request, Payment.class);
```

Customer service implementation using choreography

Implementation of customer service comprises another kind of implementation for service composition.

Client Application location

- NutruServe swagger URL: https://bzu-nutriserve.appspot.com/swagger-ui.html
- Our client application URL: http://31.168.143.199:8080/nutriServeClient
- Source code: https://github.com/rkhalyleh/nutriServeClient

Discussion

During the numerous phases of this project we have developed many items about the service oriented architecture and web services, starting from the proposal and ending in the full web services project and client project, all the topics were related in a big picture enabled us to practice practically the web services methodologies and functionalities as well as the detailed concepts.

At architectural level, we have developed a strong reliable design and were able to stick to it during the implementation phase with small changes to leverage the complexity of the application development and concentrate on the web APIs desired.

Practicing REST and SOAP was another part which was introduced in this project, and made it powerful enough to be more than one sample project, but a real application for life use even.

Non-functional requirements and specifications of the web services was designed and monitored during this project causing to practice this level and realize those quality factors.

Service composition is another area which was covered in our project which gave us deep insight on the practical side of it.

This project comprised a challenge which enabled us to go deep enough in the web services technologies.

IMS system was built as a service oriented architecture and developed and implemented incrementally applying much concepts of SOA and web services specifications.

Conclusion

Service oriented architecture can empower the applications, and give them high values, web services as a realization of SOA, can give much information, knowledge and practice of the SOA concept if applied correctly, IMS project comprises a challenge example of web services and SOA, we have made a full practice of the concepts during our work on the project gaining a knowledge and practical experience by our collaborative work on this project.

Client project of Nutrition system comprises a good practice and experience in consuming web services and communicating with other systems and APIs, either from our IMS application on need or as a standalone full client application.

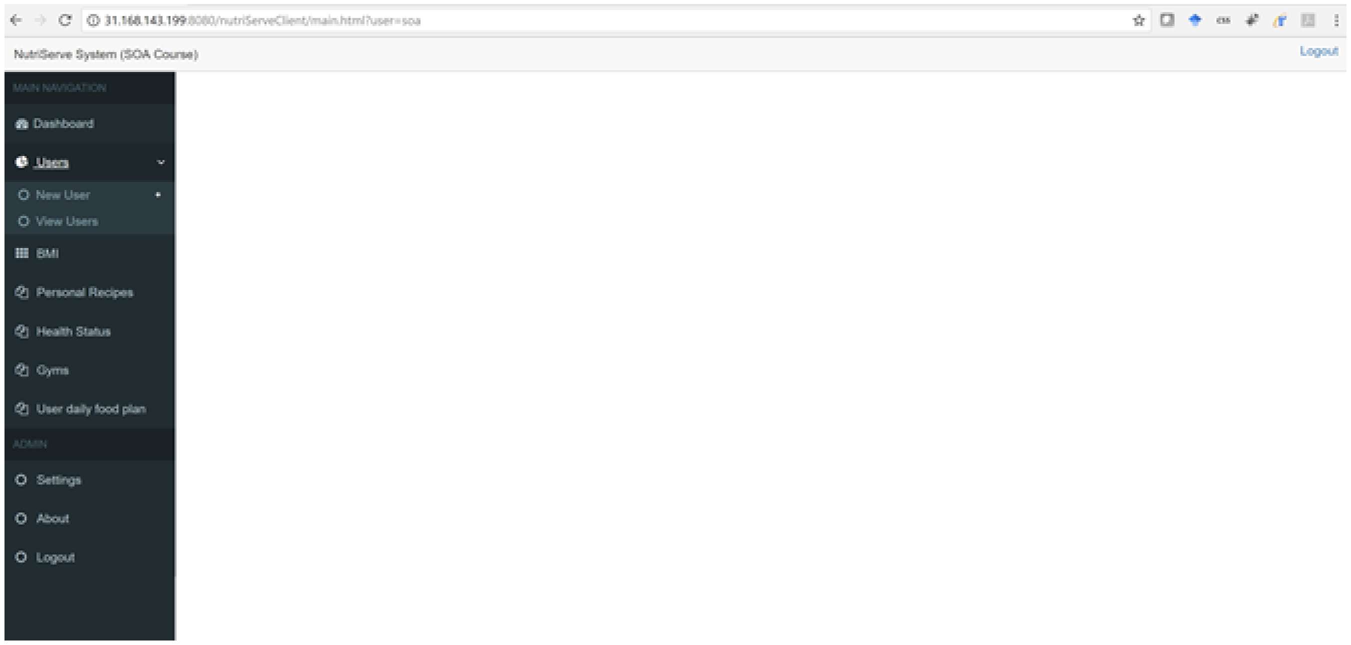
Practicing cloud service is another advantage which supports the learnability metric from this project

Appendices

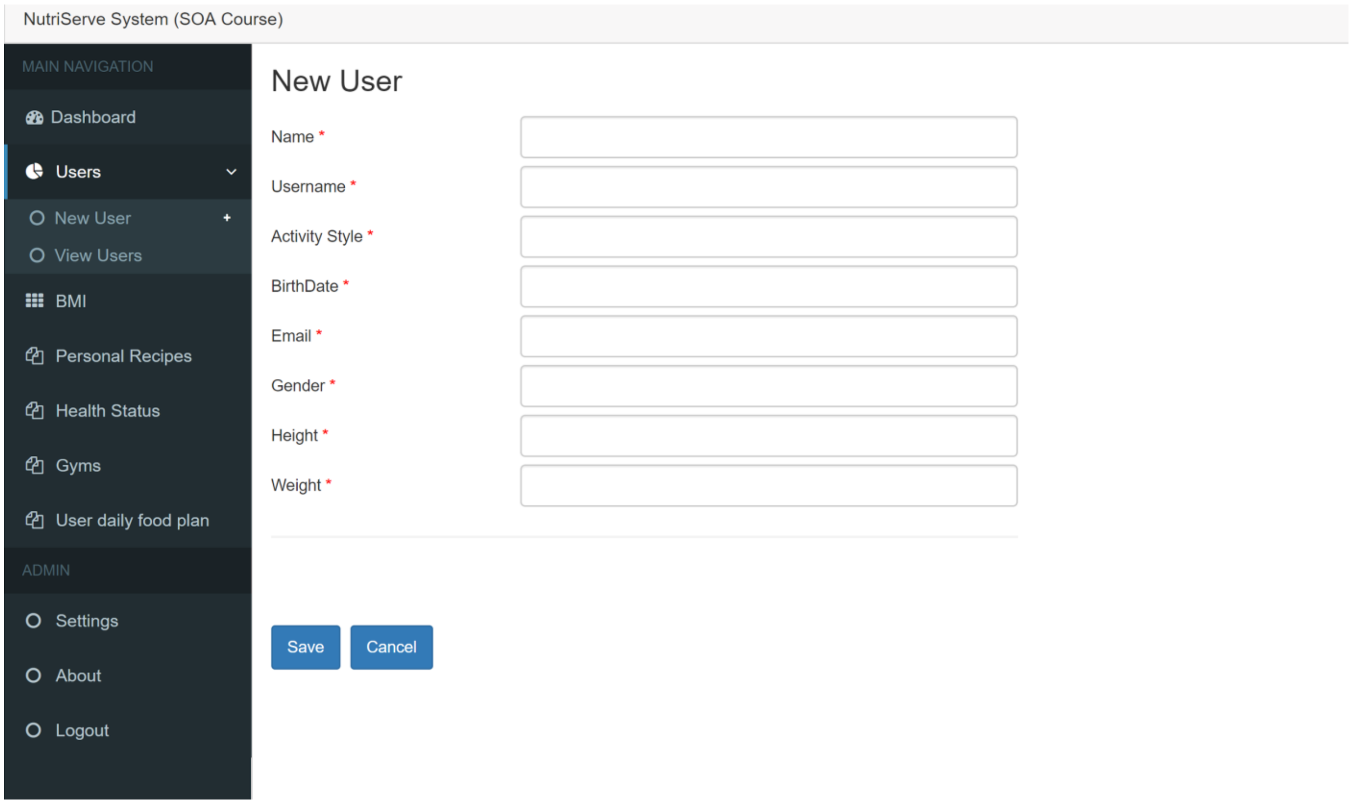
Appendix A

Client application for “NutriServe” Screenshots

- Application client main pages:



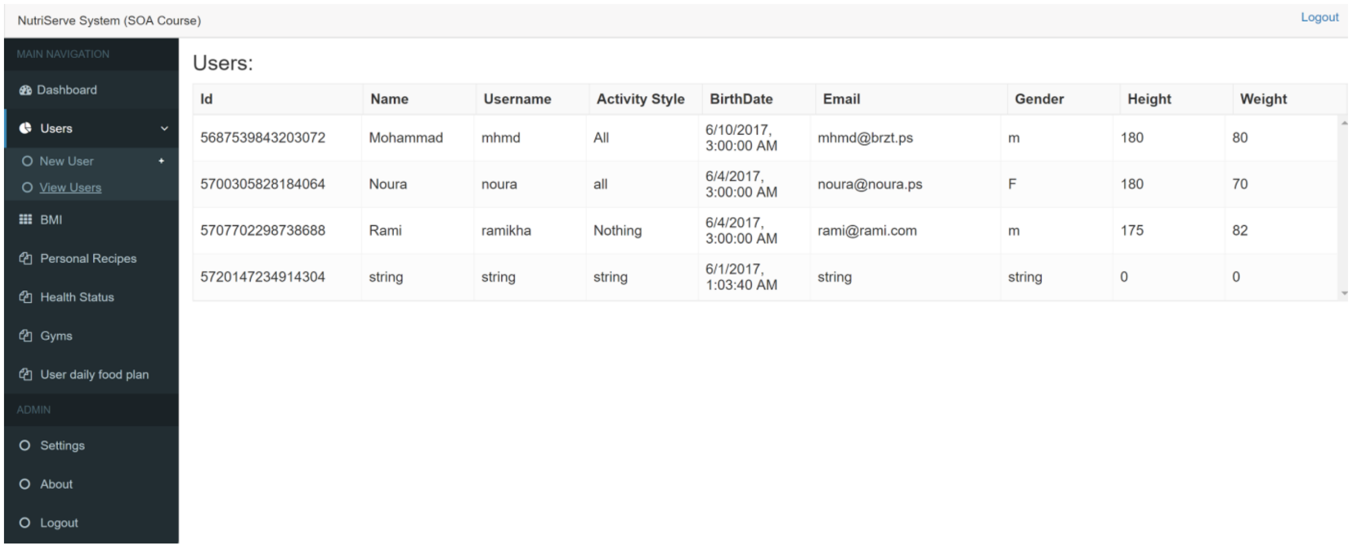
- Add new user:



New User

Name *	
Username *	
Activity Style *	
BirthDate *	
Email *	
Gender *	
Height *	
Weight *	

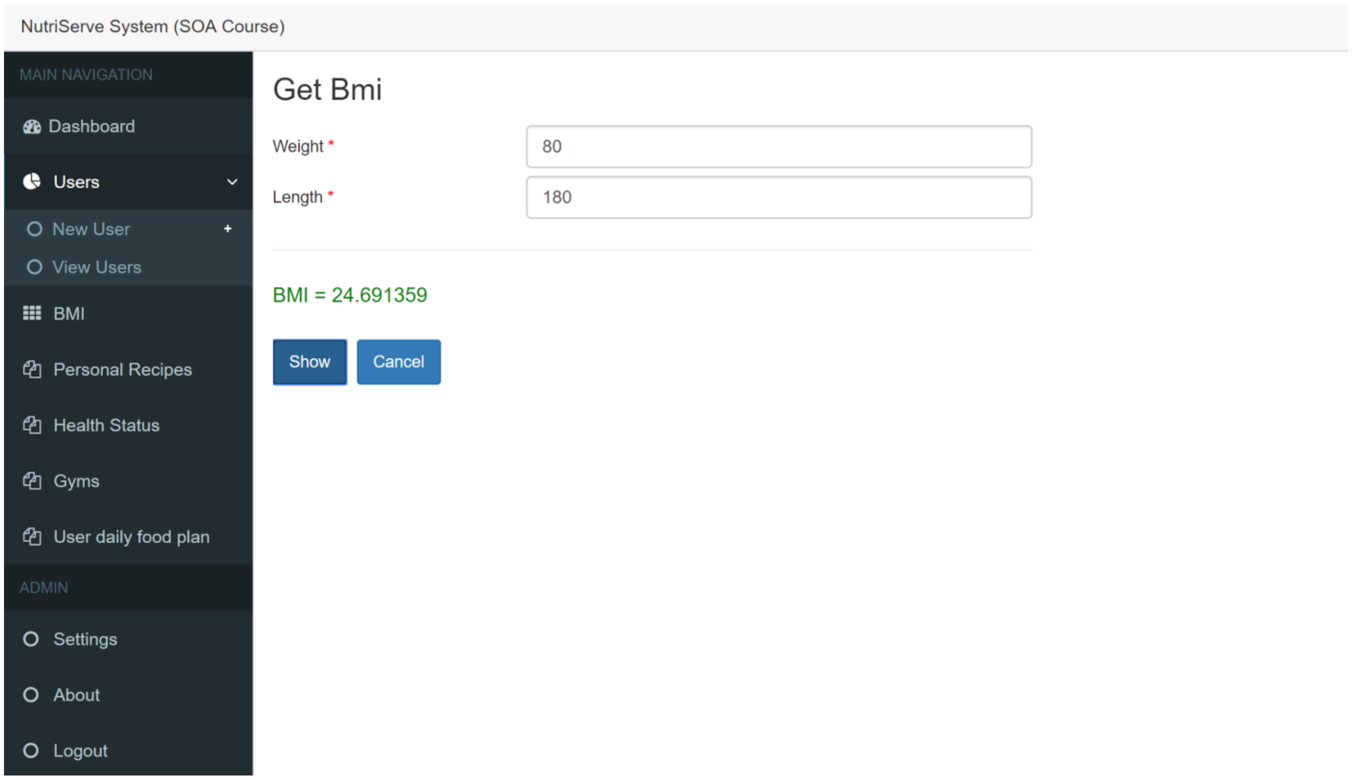
- View all Users:



Users:

Id	Name	Username	Activity Style	BirthDate	Email	Gender	Height	Weight
5687539843203072	Mohammad	mhmd	All	6/10/2017, 3:00:00 AM	mhmd@brzt.ps	m	180	80
5700305828184064	Noura	noura	all	6/4/2017, 3:00:00 AM	noura@noura.ps	F	180	70
5707702298738688	Rami	ramikha	Nothing	6/4/2017, 3:00:00 AM	rami@rami.com	m	175	82
5720147234914304	string	string	string	6/1/2017, 1:03:40 AM	string	string	0	0

- BMI

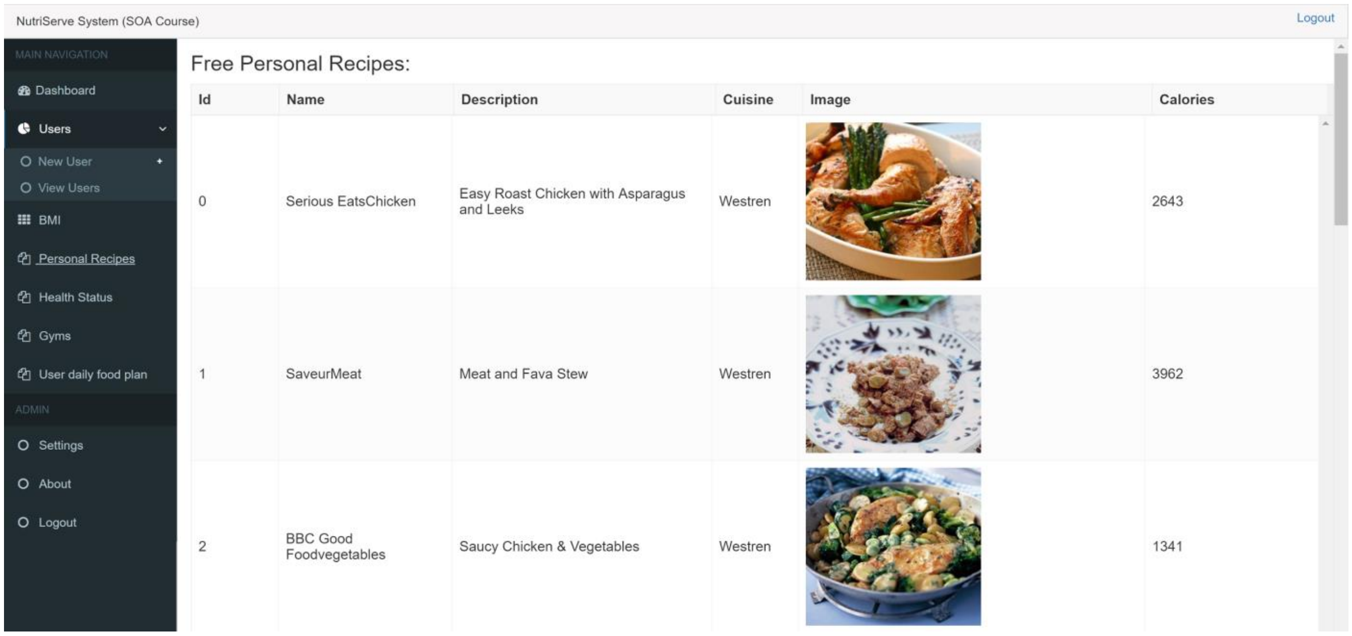


Get Bmi

Weight *	80
Length *	180

BMI = 24.691359

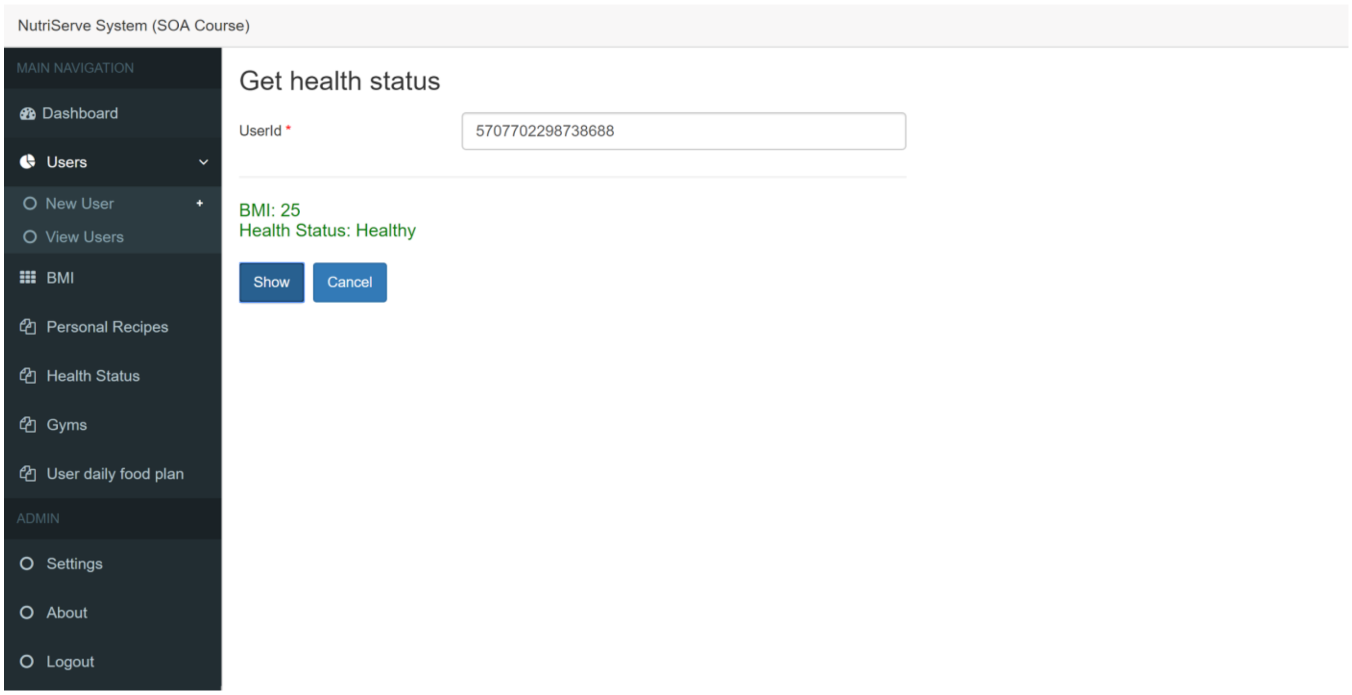
- Personal Recipes



Free Personal Recipes:

Id	Name	Description	Cuisine	Image	Calories
0	Serious EatsChicken	Easy Roast Chicken with Asparagus and Leeks	Westren		2643
1	SaveurMeat	Meat and Fava Stew	Westren		3962
2	BBC Good Foodvegetables	Saucy Chicken & Vegetables	Westren		1341

- Health status

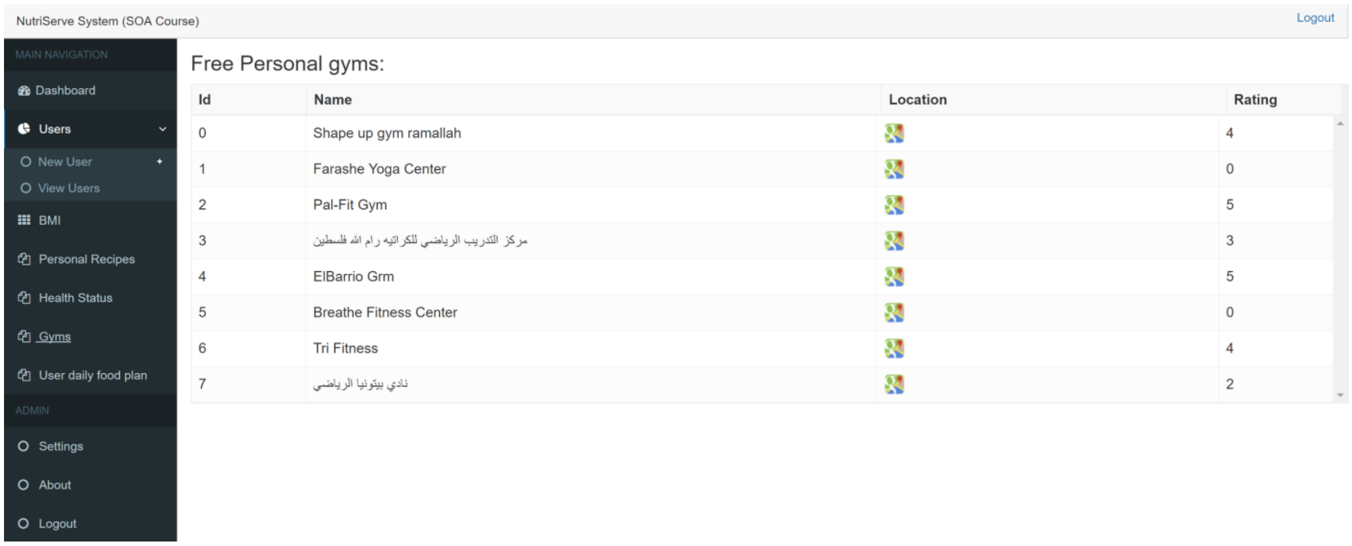


Get health status

UserId * 5707702298738688

BMI: 25
Health Status: Healthy

- Gyms



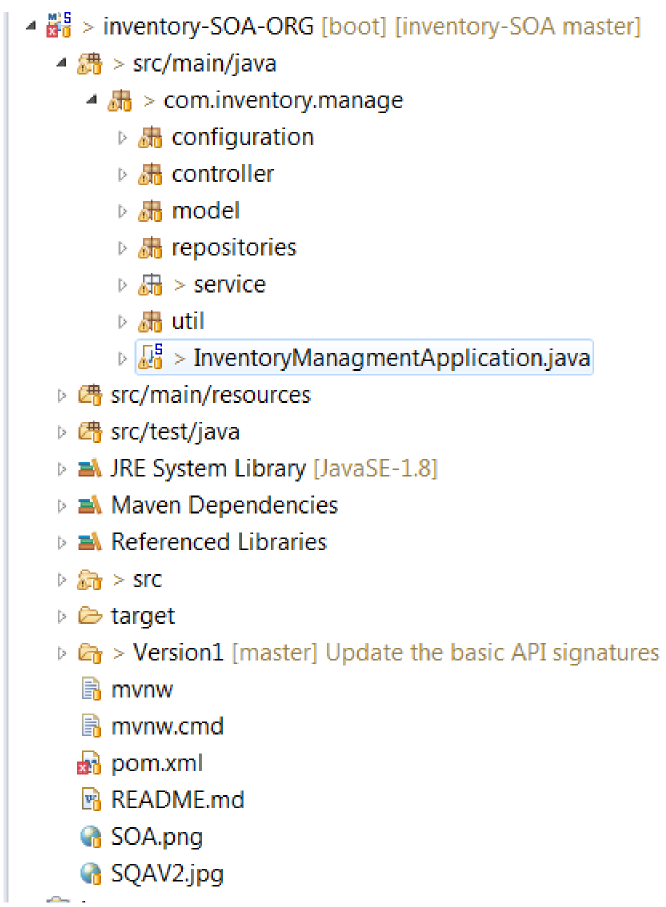
Free Personal gyms:

Id	Name	Location	Rating
0	Shape up gym ramallah	🗺	4
1	Farashe Yoga Center	🗺	0
2	Pal-Fit Gym	🗺	5
3	مركز التدريب الرياضي للكراتيه رام الله فلسطين	🗺	3
4	ElBarrio Grm	🗺	5
5	Breathe Fitness Center	🗺	0
6	Tri Fitness	🗺	4
7	نادي بيتونيا الرياضي	🗺	2

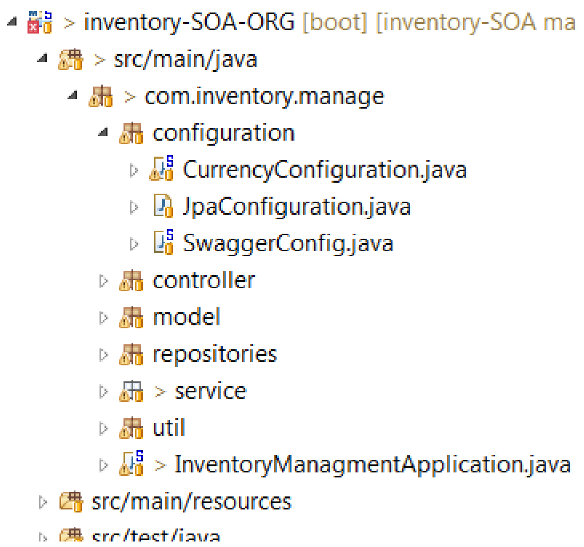
Appendix B

Web Service IMS system

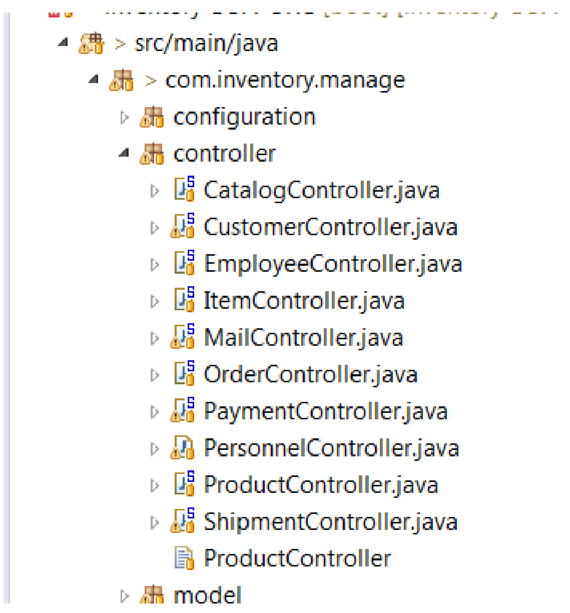
- **Application Packaging screenshots**



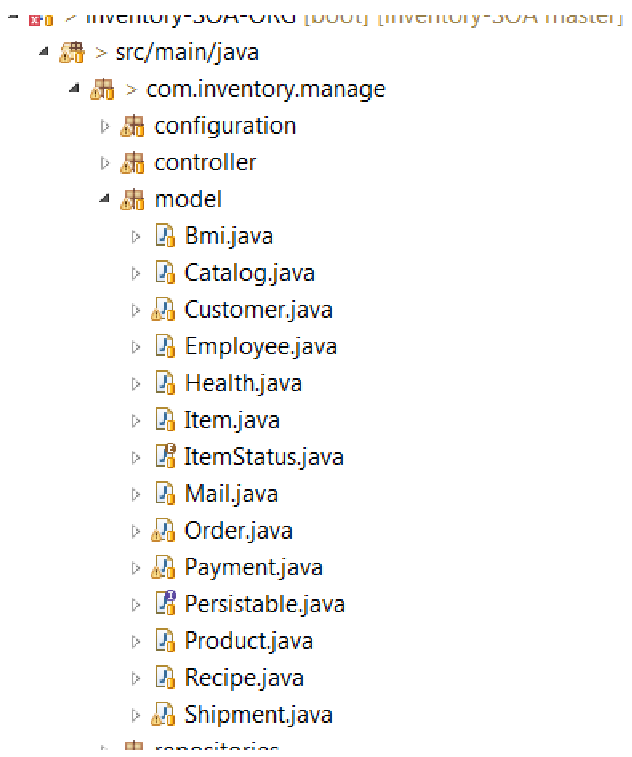
```
◢ > inventory-SOA-ORG [boot] [inventory-SOA master]
    ◢ > src/main/java
        ◢ > com.inventory.manage
            ▷ configuration
            ▷ controller
            ▷ model
            ▷ repositories
            ▷ > service
            ▷ util
            ▷ > InventoryManagmentApplication.java
    ▷ src/main/resources
    ▷ src/test/java
    ▷ JRE System Library [JavaSE-1.8]
    ▷ Maven Dependencies
    ▷ Referenced Libraries
    ▷ > src
    ▷ target
    ▷ > Version1 [master] Update the basic API signatures
      mvnw
      mvnw.cmd
      pom.xml
      README.md
      SOA.png
      SQAV2.jpg
```



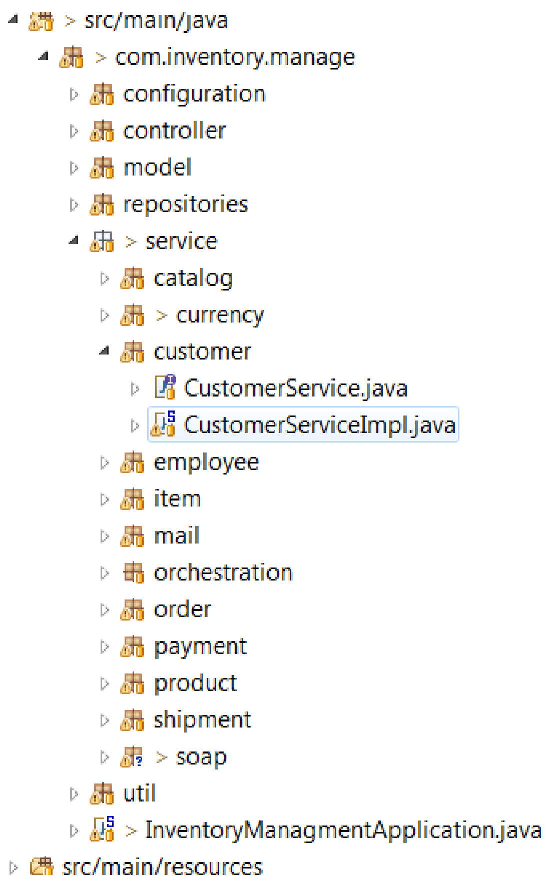
```
◢ > inventory-SOA-ORG [boot] [inventory-SOA ma
    ◢ > src/main/java
        ◢ > com.inventory.manage
            ◢ configuration
                ▷ CurrencyConfiguration.java
                ▷ JpaConfiguration.java
                ▷ SwaggerConfig.java
            ▷ controller
            ▷ model
            ▷ repositories
            ▷ > service
            ▷ util
            ▷ > InventoryManagmentApplication.java
    ▷ src/main/resources
    ▷ src/test/java
```



```
◢ > src/main/java
  ◢ > com.inventory.manage
    ▷ configuration
    ◢ controller
      ▷ CatalogController.java
      ▷ CustomerController.java
      ▷ EmployeeController.java
      ▷ ItemController.java
      ▷ MailController.java
      ▷ OrderController.java
      ▷ PaymentController.java
      ▷ PersonnelController.java
      ▷ ProductController.java
      ▷ ShipmentController.java
        ProductController
    ▷ model
```

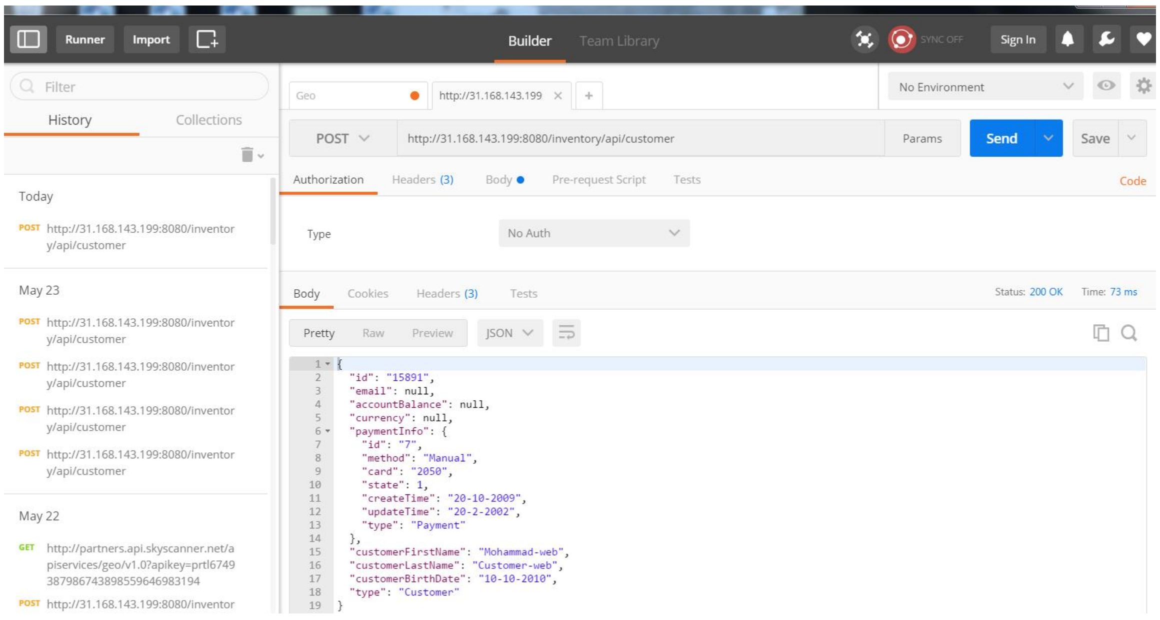


```
◢ > inventory-SOA-ORG [boot] [inventory-SOA master]
  ◢ > src/main/java
    ◢ > com.inventory.manage
      ▷ configuration
      ▷ controller
      ◢ model
        ▷ Bmi.java
        ▷ Catalog.java
        ▷ Customer.java
        ▷ Employee.java
        ▷ Health.java
        ▷ Item.java
        ▷ ItemStatus.java
        ▷ Mail.java
        ▷ Order.java
        ▷ Payment.java
        ▷ Persistable.java
        ▷ Product.java
        ▷ Recipe.java
        ▷ Shipment.java
```



```
src/main/java
  com.inventory.manage
    configuration
    controller
    model
    repositories
    service
      catalog
      currency
      customer
        CustomerService.java
        CustomerServiceImpl.java
      employee
      item
      mail
      orchestration
      order
      payment
      product
      shipment
      soap
    util
    InventoryManagmentApplication.java
src/main/resources
```

- **Postman API testing**



Today

POST http://31.168.143.199:8080/inventory/api/customer

May 23

POST http://31.168.143.199:8080/inventory/api/customer

POST http://31.168.143.199:8080/inventory/api/customer

POST http://31.168.143.199:8080/inventory/api/customer

POST http://31.168.143.199:8080/inventory/api/customer

May 22

GET http://partners.api.skyscanner.net/apiservices/geo/v1.0?apikey=prtl6749387986743898559646983194

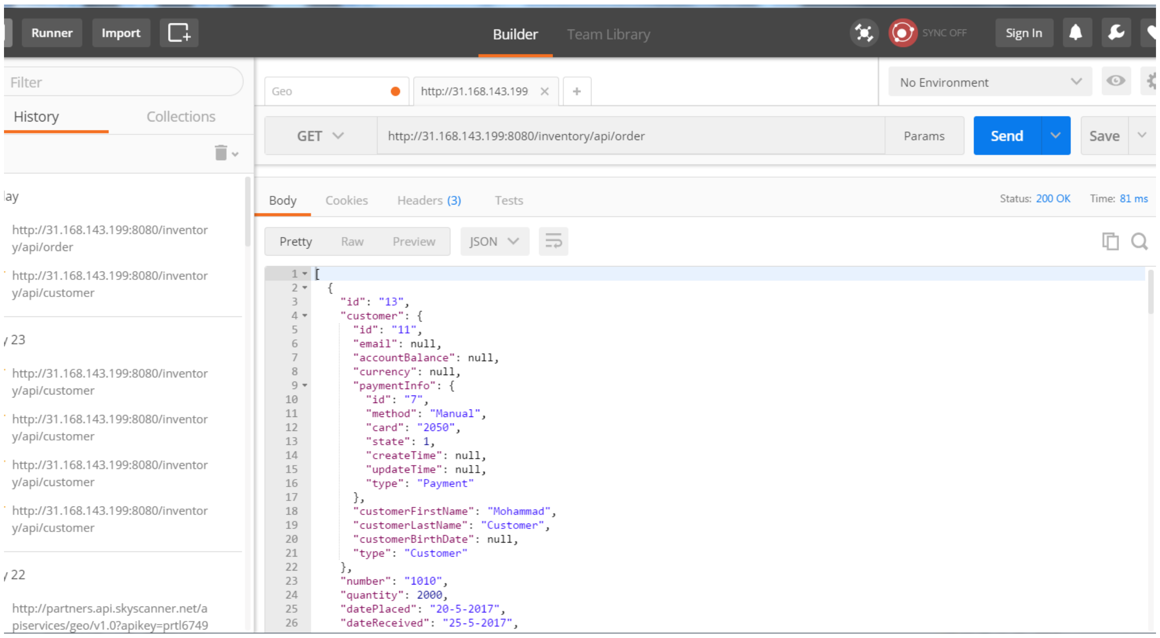
POST http://31.168.143.199:8080/inventor

POST http://31.168.143.199:8080/inventory/api/customer

Type: No Auth

Status: 200 OK Time: 73 ms

```
{
  "id": "15891",
  "email": null,
  "accountBalance": null,
  "currency": null,
  "paymentInfo": {
    "id": "7",
    "method": "Manual",
    "card": "2050",
    "state": 1,
    "createTime": "20-10-2009",
    "updateTime": "20-2-2002",
    "type": "Payment"
  },
  "customerFirstName": "Mohammad-web",
  "customerLastName": "Customer-web",
  "customerBirthDate": "10-10-2010",
  "type": "Customer"
}
```



GET http://31.168.143.199:8080/inventory/api/order

Status: 200 OK Time: 81 ms

lay

http://31.168.143.199:8080/inventory/api/order

http://31.168.143.199:8080/inventory/api/customer

/ 23

http://31.168.143.199:8080/inventory/api/customer

http://31.168.143.199:8080/inventory/api/customer

http://31.168.143.199:8080/inventory/api/customer

http://31.168.143.199:8080/inventory/api/customer

/ 22

http://partners.api.skyscanner.net/apiservices/geo/v1.0?apikey=prtl6749

```
[
  {
    "id": "13",
    "customer": {
      "id": "11",
      "email": null,
      "accountBalance": null,
      "currency": null,
      "paymentInfo": {
        "id": "7",
        "method": "Manual",
        "card": "2050",
        "state": 1,
        "createTime": null,
        "updateTime": null,
        "type": "Payment"
      },
      "customerFirstName": "Mohammad",
      "customerLastName": "Customer",
      "customerBirthDate": null,
      "type": "Customer"
    },
    "number": "1010",
    "quantity": 2000,
    "datePlaced": "20-5-2017",
    "dateReceived": "25-5-2017",
```