
Software Requirements Specification

FOR

Online Shopping API

Version 1.0 approved

Prepared by Ted Bear

Table of Contents

Contents

Table of Contents.....	2
Revision History.....	3
1. Introduction.....	4
1.1 Purpose.....	4
1.2 Intended Audience and Reading Suggestions.....	4
1.3 Product Scope.....	5
1.4 References.....	5
2. Overall Description.....	6
2.1 Product Perspective.....	6
2.2 Product Functions.....	6
2.3 Running the Application.....	22
2.4 Operating Environment.....	23
2.5 Design and Implementation Constraints.....	23
2.6 Assumptions and Dependencies.....	23
3. External Interface Requirements.....	24
3.1 Hardware Interfaces.....	24
3.2 Software Interfaces.....	24
3.3 Communications Interfaces.....	24
4. Nonfunctional Requirements.....	25
4.1 Performance Requirements.....	25
4.2 Safety Requirements.....	25
4.3 Security Requirements.....	25
4.4 Software Quality Attributes.....	25

Revision History

Name	Date	Reason for Changes	Version	
Ted Bear	5/22/2020	Initial project feature	1.0	

1. Introduction

1.1 Purpose

This document is meant to delineate the features of OSS, so as to serve as a guide to the developers on one hand and a software validation document for the prospective client on the other. The Online Shopping System (OSS) for multiple items shop web application is intended to provide complete solutions for vendors as well as customers through a single gateway using the internet.

It will enable vendors to setup online shops, customer to browse through the shop and purchase them online without having to visit the shop physically. The administration module will enable a system administrator to approve and reject requests for new categories and maintain various lists of categories.

1.2 Intended Audience and Reading Suggestions

This document is to be read by the development team, the project manager, marketing staff, testers and documentation writers. Our stakeholders may review the document to learn about the project and to understand the requirements. The SRS has been organized approximately in order of increasing specificity. The developers and project managers need to become intimately familiar with the SRS.

Others involved need to review the document as such:

Overall Description – Marketing staff must become accustomed to the various product features in order to effectively advertise the product.

System features – Testers need an understanding of the system features to develop meaningful test cases and give useful feedback to the developers.

The author would suggest clients to go through the requirement section thoroughly before installing the software. The testers are expected to have certain knowledge in the terms used and hence can go for the security issues directly. developers can utilize the documentation as a resource in developing the project to a new product.

1.3 Product Scope

The main goal of the application is to maintain and manage the users, bills, products, orders and every other aspect of online shopping. Online shopping Store Management software is very needed for Online Shopping Store. This software helps them maintain day to day transactions in computer.

1.4 References

https://en.wikipedia.org/wiki/Online_shopping
<https://icons8.com/articles/e-commerce-features-successful-online-store/>

2. Overall Description

2.1 Product Perspective

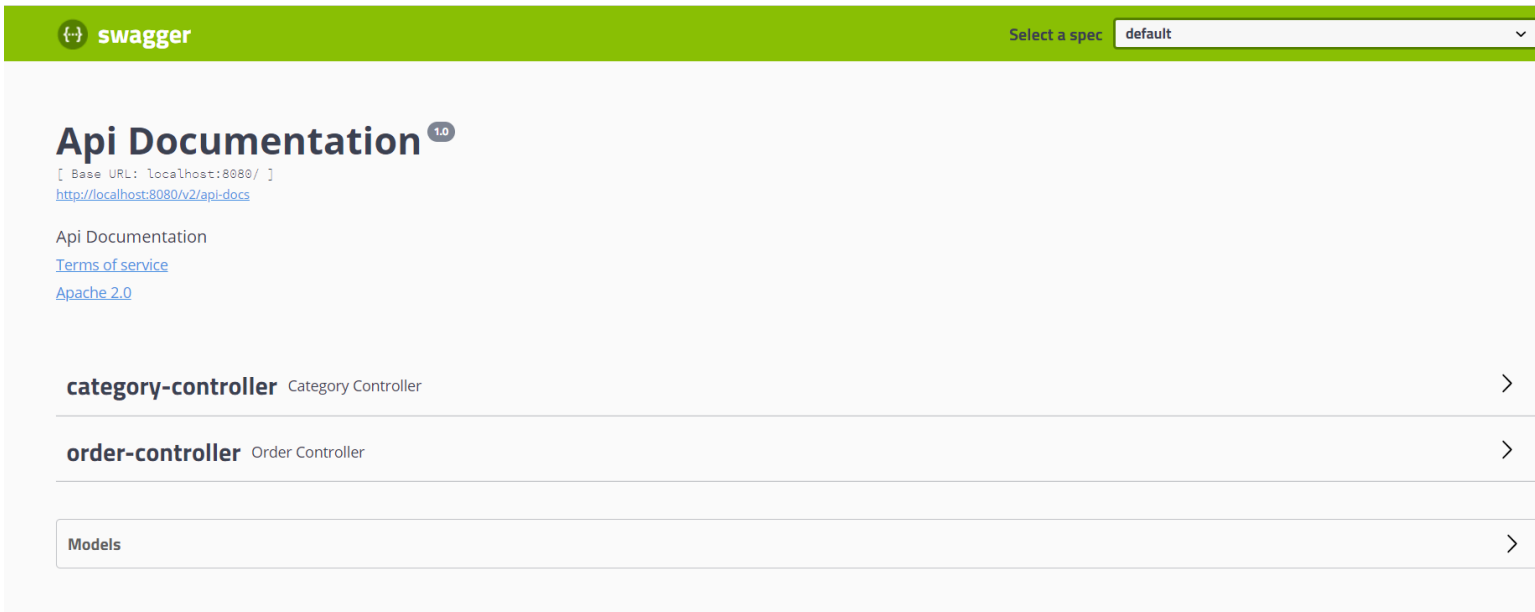
Main goal of the application is to maintain the records of Order, Sales and stock details with Bill transaction maintenance.

2.2 Product Functions

Craft provides the following described features: -

This project consists of REST and SOAP services that manages getting product details, order details, and user detail. It provides services for managing products and orders. Products can be configured with rate and category.

REST services:



The image shows the Swagger API Documentation interface. At the top, there is a green header bar with the Swagger logo on the left and a dropdown menu labeled "Select a spec" with "default" selected on the right. Below the header, the main content area has a light gray background. It starts with the title "Api Documentation" followed by a version badge "1.0". Below the title, there is a small text block containing the base URL "[Base URL: localhost:8080/]" and a link "http://localhost:8080/v2/api-docs". Underneath, there are three links: "Api Documentation", "Terms of service", and "Apache 2.0". The main section lists two API endpoints: "category-controller" with the description "Category Controller" and "order-controller" with the description "Order Controller". Each endpoint has a right-pointing chevron icon. At the bottom, there is a section titled "Models" with a right-pointing chevron icon.

swagger

Select a spec default

Api Documentation ^{1.0}

[Base URL: localhost:8080/]
<http://localhost:8080/v2/api-docs>

Api Documentation
[Terms of service](#)
[Apache 2.0](#)

category-controller Category Controller >

order-controller Order Controller >

Models >

Category-controller:

REST call – **findAllCategories()**

category-controller

Category Controller

▼

GET

/categories

findAllCategories

Parameters

Cancel

No parameters

Execute

Clear

Responses

Response content type

/

▼

Curl

curl -X GET "http://localhost:8080/categories" -H "accept: */*"

Request URL

http://localhost:8080/categories

Server response

Code

Details

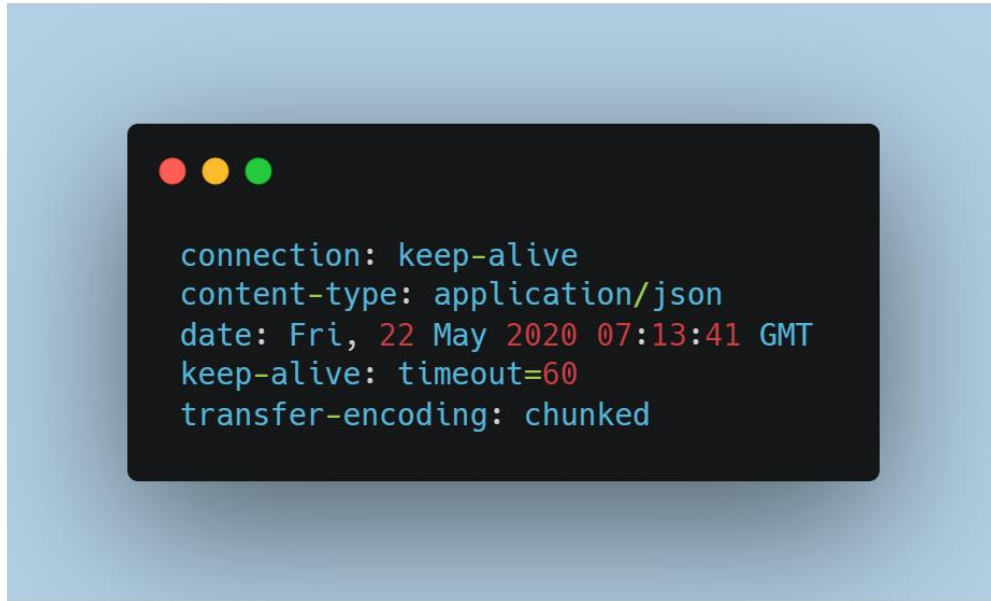
200

Response body

Response Body:


```
[
  {
    "categoryid": 11,
    "name": "laptops"
  },
  {
    "categoryid": 12,
    "name": "home electronics"
  },
  {
    "categoryid": 13,
    "name": "mouse"
  },
  {
    "categoryid": 14,
    "name": "mobile accesories"
  },
  {
    "categoryid": 15,
    "name": "laptop skins"
  },
  {
    "categoryid": 16,
    "name": "laptop covers"
  },
  {
    "categoryid": 17,
    "name": "men's clothing"
  },
  {
    "categoryid": 18,
    "name": "women' wardrobe"
  },
  {
    "categoryid": 19,
    "name": "gaming geeks"
  },
  {
    "categoryid": 20,
    "name": "study buddy"
  }
]
```

Response header:



Responses:

Responses	
Code	Description
200	<div>OK</div> <div>Example Value Model</div> <div>[{ "categoryid": 0, "name": "string" }]</div>
401	<div>Unauthorized</div>
403	<div>Forbidden</div>
404	<div>Not Found</div>

REST call – findcategory (by Id)

GET
/category/{id} findcategory

Parameters
Cancel

Name	Description
id * required integer(\$int32) (path)	id <input type="text" value="11"/>

Execute
Clear

Responses
Response content type */*

Curl

```
curl -X GET "http://localhost:8080/category/11" -H "accept: */*"
```

Request URL

```
http://localhost:8080/category/11
```

Server response

Code	Details
------	---------

Response body:



Response header:

```

connection: keep-alive
content-type: application/json
date: Fri, 22 May 2020 07:16:21 GMT
keep-alive: timeout=60
transfer-encoding: chunked
    
```

Responses:

Responses	
Code	Description
200	<div>OK</div> <div>Example Value Model</div> <div> <pre>{ "categoryid": 0, "name": "string" }</pre> </div>
401	<div>Unauthorized</div>
403	<div>Forbidden</div>
404	<div>Not Found</div>

Order Controller:

Rest Call – getorder (by ID)

GET

/order/{id} getorder

Parameters

Cancel

Name	Description
id <small>* required</small> integer(\$int32) (path)	id <input type="text" value="51"/>

Execute

Clear

Responses

Response content type

Curl

```
curl -X GET "http://localhost:8080/order/51" -H "accept: */"
```

Request URL

```
http://localhost:8080/order/51
```

Server response

Code	Details
200	Response body

Response body:

```
{
  "orderid": 51,
  "userid": 1,
  "addressid": 201,
  "discountid": 6,
  "created": "2019-05-15 12:39:30",
  "modified": "2019-05-15 15:39:30",
  "status": "shipped",
  "amount": 550
}
```

Response headers:

```
connection: keep-alive
content-type: application/json
date: Fri, 22 May 2020 07:33:47 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses:

Responses	
Code	Description
200	<div>OK</div> <div>Example Value Model</div> <div><pre>{ "addressid": 0, "amount": 0, "created": "string", "discountid": 0, "modified": "string", "orderid": 0, "status": "string", "userid": 0 }</pre></div>
401	<div>Unauthorized</div>
403	<div>Forbidden</div>
404	<div>Not Found</div>

REST Call – getallorders()

GET

/orders getallOrders

Parameters

Cancel

No parameters

Execute

Clear

Responses

Response content type */*

Curl

curl -X GET "http://localhost:8080/orders" -H "accept: */*"

Request URL

http://localhost:8080/orders

Server response

Code	Details
200	Response body

Response body:

```
[
  {
    "orderid": 51,
    "userid": 1,
    "addressid": 201,
    "discountid": 6,
    "created": "2019-05-15 12:39:30",
    "modified": "2019-05-15 15:39:30",
    "status": "shipped",
    "amount": 550
  },
  {
    "orderid": 52,
    "userid": 2,
    "addressid": 202,
    "discountid": 8,
    "created": "2019-12-23 12:07:52",
    "modified": null,
    "status": "completed",
    "amount": 8000
  },
  {
    "orderid": 53,
    "userid": 9,
    "addressid": 209,
    "discountid": 10,
    "created": "2020-01-22 12:08:29",
    "modified": "2020-05-13 12:08:29",
    "status": "Awaiting payment",
    "amount": 952
  },
  {
    "orderid": 54,
    "userid": 5,
    "addressid": 205,
    "discountid": 4,
    "created": "2019-06-10 12:09:52",
    "modified": null,
    "status": "Awaiting shipment",
    "amount": 920
  },
  {
    "orderid": 55,
    "userid": 10,
    "addressid": 210,
    "discountid": 5,
    "created": "2019-12-10 12:10:37",
    "modified": "2019-12-12 14:35:05",
    "status": "Partially Shipped.",
    "amount": 9898
  },
]
```



```
{
  "orderid": 56,
  "userid": 6,
  "addressid": 206,
  "discountid": 2,
  "created": "2020-05-04 12:11:30",
  "modified": null,
  "status": "Cancelled",
  "amount": 8528
},
{
  "orderid": 57,
  "userid": 9,
  "addressid": 209,
  "discountid": 1,
  "created": "2020-01-13 12:12:16",
  "modified": "2020-02-28 12:12:16",
  "status": "Completed",
  "amount": 97859
},
{
  "orderid": 58,
  "userid": 8,
  "addressid": 208,
  "discountid": 7,
  "created": "2020-03-07 12:12:56",
  "modified": null,
  "status": "Completed",
  "amount": 10000
},
{
  "orderid": 59,
  "userid": 2,
  "addressid": 202,
  "discountid": 5,
  "created": "2019-12-24 12:13:39",
  "modified": null,
  "status": "Awaiting shipment",
  "amount": 89
},
{
  "orderid": 60,
  "userid": 1,
  "addressid": 201,
  "discountid": 1,
  "created": "2020-05-20 12:14:27",
  "modified": null,
  "status": "Completed",
  "amount": 98765
}
}
```

Response headers:



```
connection: keep-alive
content-type: application/json
date: Fri, 22 May 2020 08:20:05 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses:

Responses	
Code	Description
200	<div>OK</div> <div>Example Value Model</div> <div>[{ "addressid": 0, "amount": 0, "created": "string", "discountid": 0, "modified": "string", "orderid": 0, "status": "string", "userid": 0 }]</div>
401	<div>Unauthorized</div>
403	<div>Forbidden</div>
404	<div>Not Found</div>

Models:

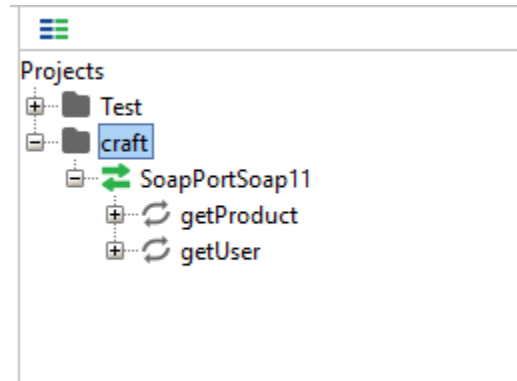
Models

```
Category ▾ {  
  categoryid      integer($int32)  
  name            string  
}
```

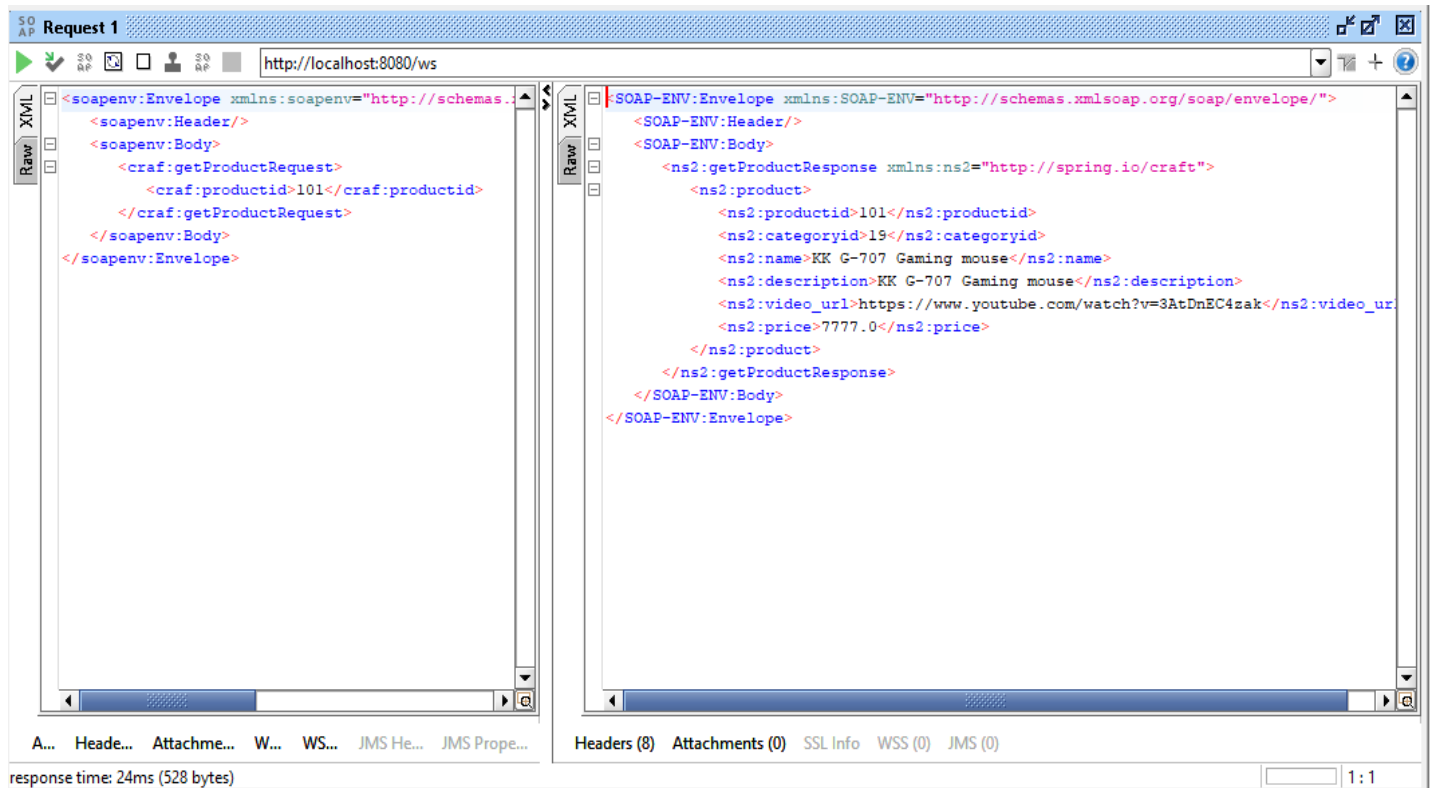
```
Order ▾ {  
  addressid      integer($int32)  
  amount         number($float)  
  created        string  
  discountid     integer($int32)  
  modified       string  
  orderid        integer($int32)  
  status         string  
  userid         integer($int32)  
}
```

SOAP web services:

Using SOAPUI to work as a SOAP client and get registered services. The initial structure of the SOAP web services looks as below: The wsdl file is accessible at “ <http://localhost:8080/ws/craft> ”.



getProduct Service:



The screenshot displays the 'Request 1' window in SOAPUI, showing the raw XML for a SOAP request and its corresponding response. The address bar indicates the URL `http://localhost:8080/ws`.

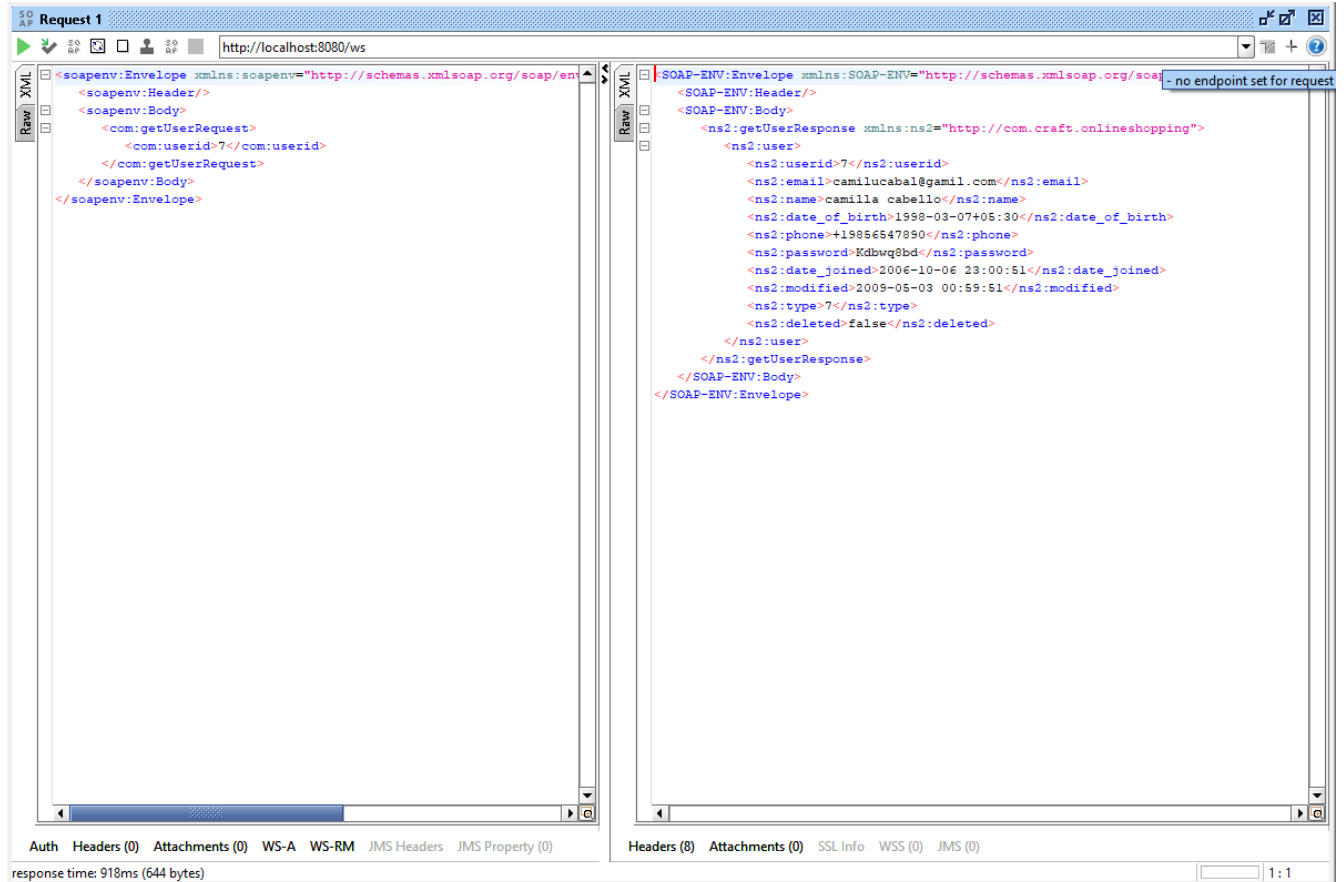
Request XML:

```
<?xml version='1.0'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <craf:getProductRequest>
      <craf:productid>101</craf:productid>
    </craf:getProductRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Response XML:

```
<?xml version='1.0'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:getProductResponse xmlns:ns2="http://spring.io/craft">
      <ns2:product>
        <ns2:productid>101</ns2:productid>
        <ns2:categoryid>19</ns2:categoryid>
        <ns2:name>KK G-707 Gaming mouse</ns2:name>
        <ns2:description>KK G-707 Gaming mouse</ns2:description>
        <ns2:video_url>https://www.youtube.com/watch?v=3AtDnEC4zak</ns2:video_url>
        <ns2:price>7777.0</ns2:price>
      </ns2:product>
    </ns2:getProductResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The status bar at the bottom shows a response time of 24ms (528 bytes) and a 1:1 ratio.

getUser Service:

The screenshot displays a SOAP client interface with two panels showing XML data. The left panel, titled "Request 1", shows the SOAP request XML. The right panel shows the SOAP response XML. The response contains user details for a user with ID 7.

Request XML:

```
<?xml version='1.0' encoding='UTF-8'>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <com:getUserRequest>
      <com:userid>7</com:userid>
    </com:getUserRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Response XML:

```
<?xml version='1.0' encoding='UTF-8'>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:getUserResponse xmlns:ns2="http://com.craft.onlineshopping">
      <ns2:user>
        <ns2:userid>7</ns2:userid>
        <ns2:email>camilucabal@gmail.com</ns2:email>
        <ns2:name>camilla cabello</ns2:name>
        <ns2:date_of_birth>1998-03-07+05:30</ns2:date_of_birth>
        <ns2:phone>+19856547890</ns2:phone>
        <ns2:password>Kdbwq8bd</ns2:password>
        <ns2:date_joined>2006-10-06 23:00:51</ns2:date_joined>
        <ns2:modified>2009-05-03 00:59:51</ns2:modified>
        <ns2:type>7</ns2:type>
        <ns2:deleted>false</ns2:deleted>
      </ns2:user>
    </ns2:getUserResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The interface also shows a status bar at the bottom with the text "response time: 918ms (644 bytes)".

2.3 Running the Application

This application has been using Spring Boot as it provides a wide variety of features that aid development and maintenance.

Some features that were utilized were: Spring Security, Spring Data/JPA and starters. SOAP services need a SOAP client to verify. Here, we have used SOAPUI. Any SOAP client UI can be used. This program and instructions have been tested on following versions on Windows laptop.

- Apache Maven 3.5.0
- Java version: 1.8.0_131
- MySQL: 8.0.20
- SOAPUI : 5.5.0

How to run the application locally?

Pre-requisites to run application are Java, Maven and Git.

Installation instructions for Maven are available at <https://maven.apache.org/install.html>

Java can be installed from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

MYSQL can be installed from <https://dev.mysql.com/downloads/mysql/>

SOAPUI can be installed from <https://www.soapui.org/downloads/latest-release/>

Steps:

1. Open Command Line Interface
2. Cd to <jar file path>
3. Type `java -jar REST-online shopping-0.0.1-SNAPSHOT.jar --server.port=8080`

2.4 Operating Environment

This site is compatible with almost operating system, to mention some:

- Window OS
- Linux OS
- Mac OS

2.5 Design and Implementation Constraints

2.5.1 Government Policies: Any policy made to regulate content of discussion or any type of censorship imposed on the site could cause inefficient function of the site.

1.1.1 - Attacks: Any cyber-attack, specifically distributed denial of services could cause loss financially as well as to the productivity of the site. Proper resources and pre-planning need to be implanted to face the same.

1.1.2 - Post Content: Any improper or vulgar post could harm the reputation of the site as it is viewed by plenty of Orders including various age groups with class.

2.6 Assumptions and Dependencies

It is assumed that the software designed will work correctly with Linux, Windows or Mac Operating System. With the given hardware requirements, software will work efficiently.

3. External Interface Requirements

3.1 Hardware Interfaces

This product requires a system with the following requirements:

Intel Xeon E9 Quad Core Processor clocked at 4GHz with liquid cooling

features ADATA 128GB Dual Channel DDR4 Ram RAID10 100TB Storage

Optical Fibers connecting every machine to the routers

External Cooling System for the CPU as well as hard disk

3.2 Software Interfaces

The following software needs to be installed in the system:

- JDK 1.7 and plus.
- Maven
- SOAPUI

3.3 Communications Interfaces

This product requires HTTP protocols to be enabled in the firewall or iptables.

A network connectivity with more than or equal to 10Mbps bandwidth is required.

4. Nonfunctional Requirements

4.1 Performance Requirements

The performance of the product under the mentioned hardware and software requirements is expected to result good. Upgrading the system, especially hard drive and needs, network connectivity and cooling system could effectively improvise results.

4.2 Safety Requirements

All the mentioned protocols need to be enabled and expected to be properly working on respective ports.

Failure of any hardware or software dependencies could lead to malfunction or downtime for the site.

Prevention for various cyber-attacks should be initiated by default as well as additional manual monitoring should be engaged to prevent downtime.

4.3 Security Requirements

A dedicated cyber-security team should be engaged to monitor attacks and prevent downtime.

Using obsolete software should be avoided and there should be timely check for updates.

Depending on the number of Orders, hardware should be upgraded. There should also be timely check for maintenance of components such as hard drive, mother board etc.

4.4 Software Quality Attributes

This product is applicable to age groups above 3. This will be available in all regions except Germany and Poland.

Online shopping API is written in object-oriented pattern and has Restful architecture. Hence it sports easy portability as well as scalability.

Regular backup is taken and synchronized to prevent any type of loss.