**Technologies used to implement**:

**Server-side Technologies:** Used Java 8, Spring Boot & Maven. Use of Spring Boot will give you a standalone, fully shippable application which you can extract and run on any system where Java and Maven are installed. Used Junit and Mockito for server side testing. JSON for request and response of the APIs.

RESTful APIs can be separately tested using any REST client. (Screenshots attached for reference.)

**UI Technologies:** Used bootstrap css, jquery, knockout js and html for the implementation of UI.

**Notes and Background:**

Provided “channelsubscription” project. It is mainly implemented using Java 8 , Spring boot. Spring Boot provides the facility to run the project as **standalone** component which uses Spring Boot embedded tomcat.

Spring boot uses the **8080** port of local machine. This 8080 port can be configured in the application.properties. Current project uses the default 8080 port

Before testing few assumption are made to fulfil each step in the requirement:

1. As there is no database used, database in memory stub is used return data.

Valid customerIds are “1001” and “1002”

Except all others are considered as the customers which doesn’t have data and system will throw error as below:

"There was a problem retrieving the customer information"

1. As mentioned in the requirement Product Categories for locationIds are used:

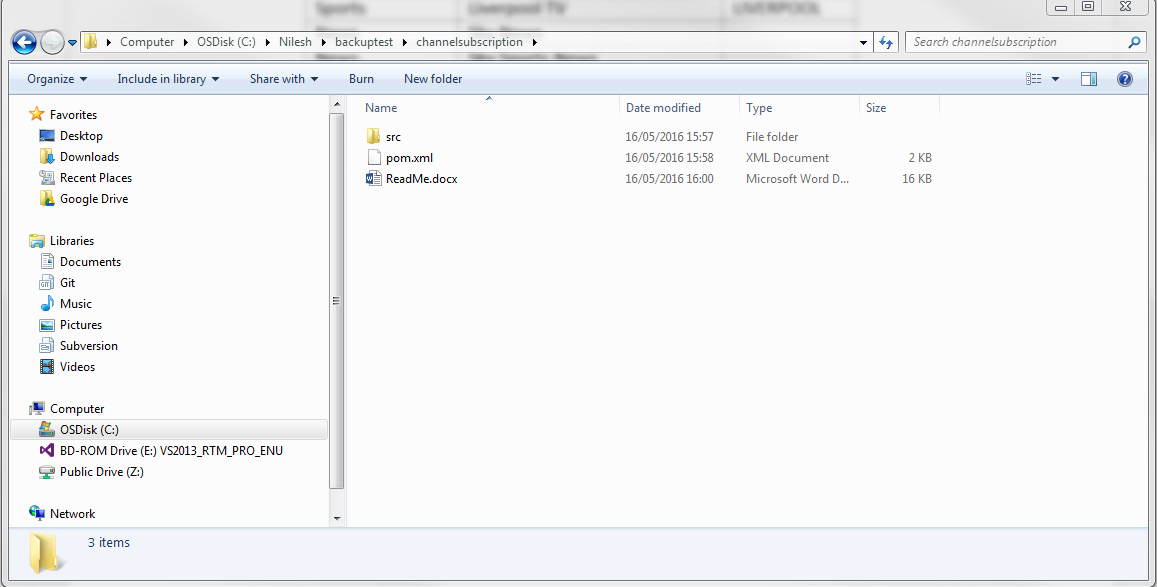
|  |  |  |
| --- | --- | --- |
| Category | Product | LocationId |
| Sports | Arsenal TV | LONDON |
| Sports | Chelsea TV | LONDON |
| Sports | Liverpool TV | LIVERPOOL |
| News | Sky News |  |
| News | Sky Sports News |  |

So only above locationIds should be used.

**How to Setup and test:**

**Prerequisite: Java 8 installed on the machine, Maven 3.x step on machine where the project is run and any RESTClient should be available to test APIs only.**

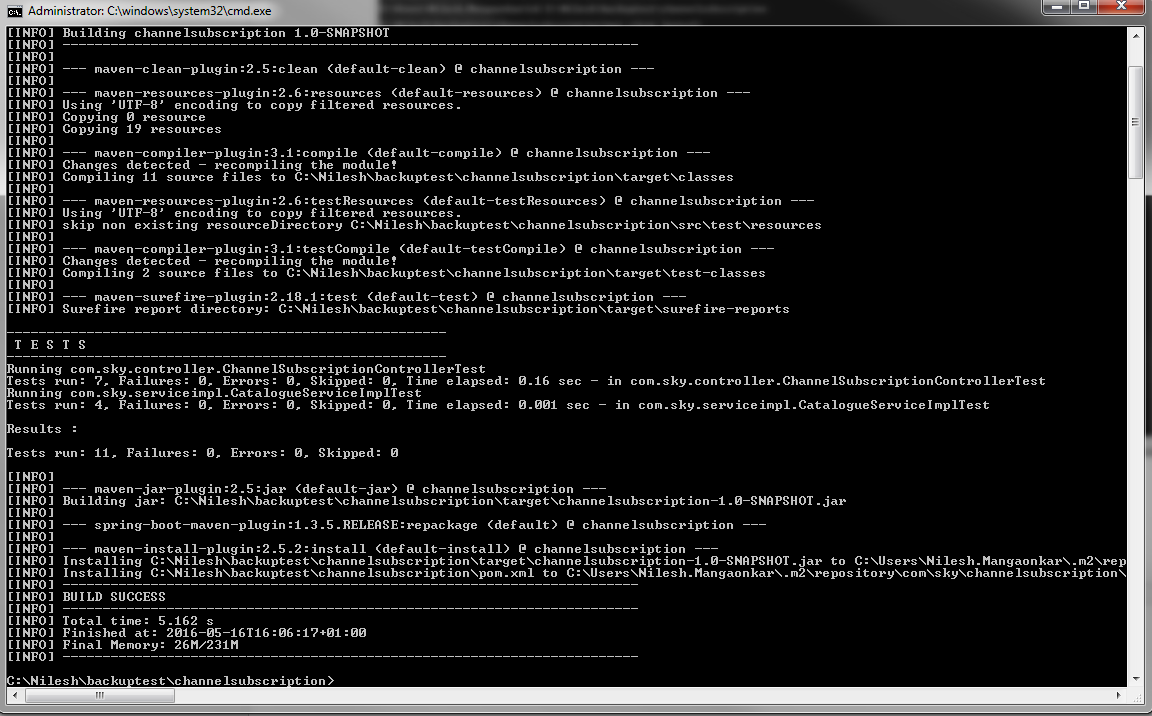
1. Copy the “channelsubscription.zip” to any suitable location and extract it to “channelsubscription” Shown screenshot as below:



1. Go the the project location on command prompt and run command “**mvn clean install**” as shown below:



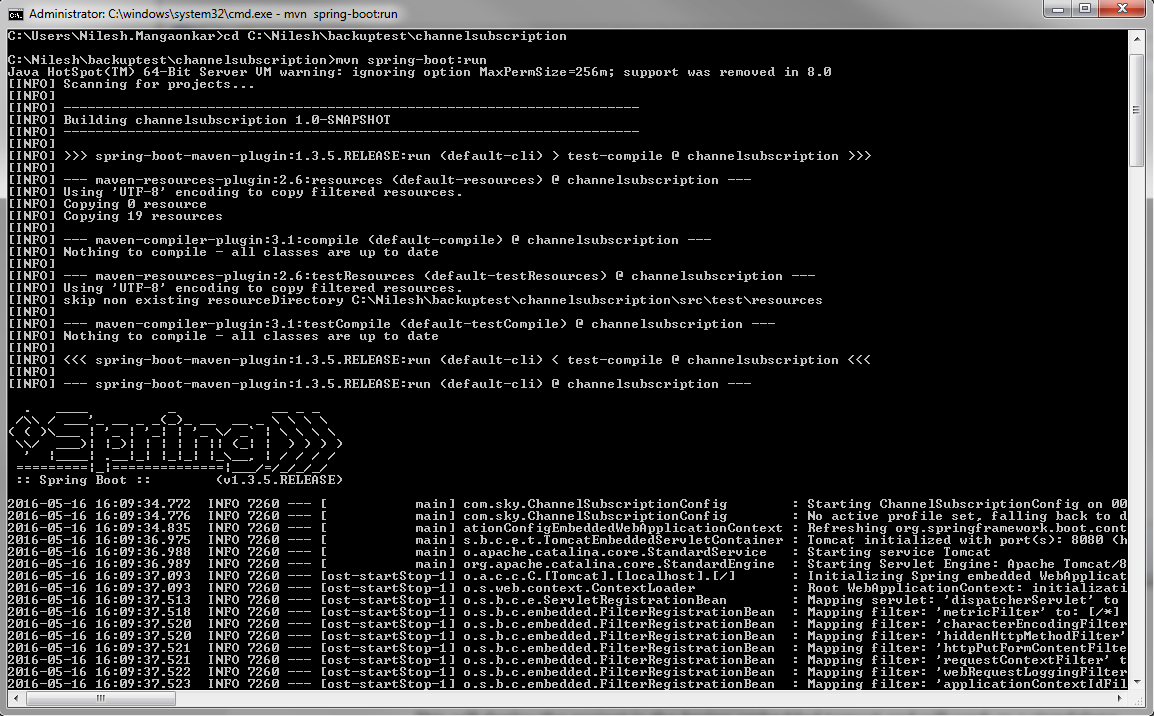
1. You see the build successful as below:



1. Now the project can be run using below command of spring-boot:

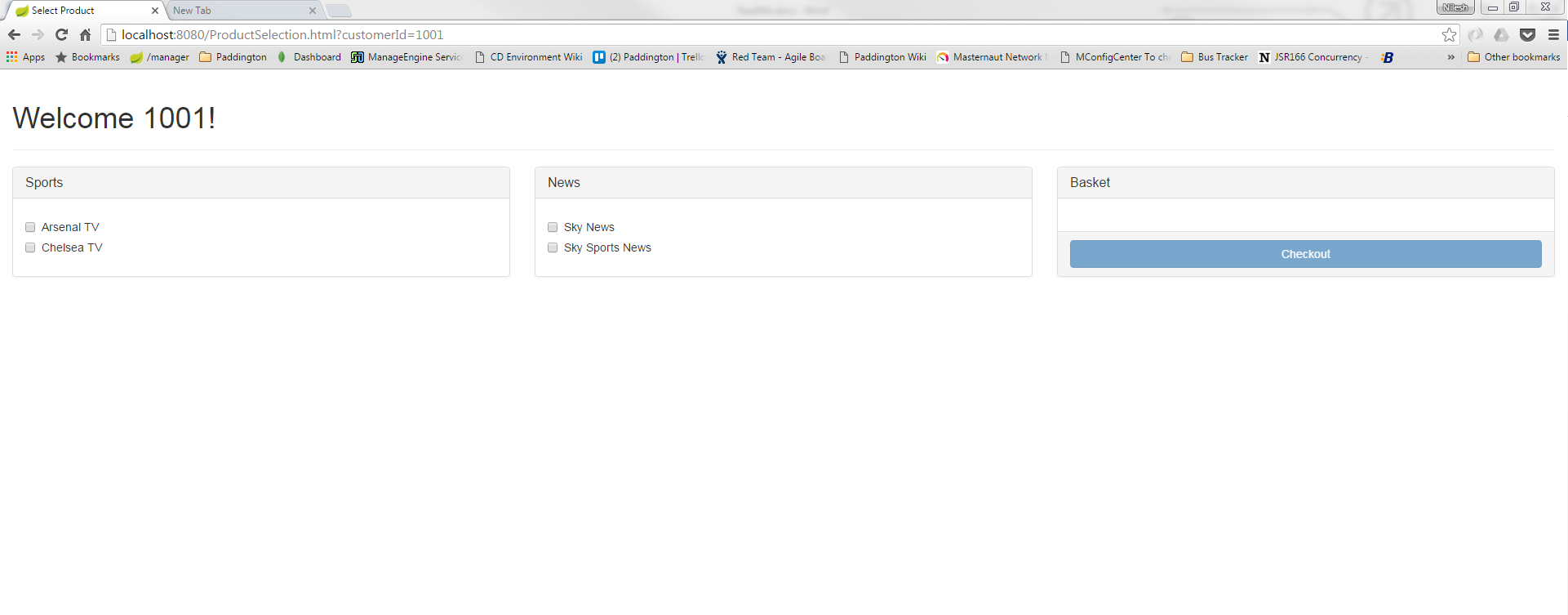
**mvn spring-boot:run**

This will deploy the project in the Spring embedded tomcat and will work as a standalone component. See below screenshot



1. The application can be run using below url in any browser:

<http://localhost:8080/ProductSelection.html?customerId=1001>



**Note: The customerId is sent as part of URL in this case and this URL can be configured in any application when required.**

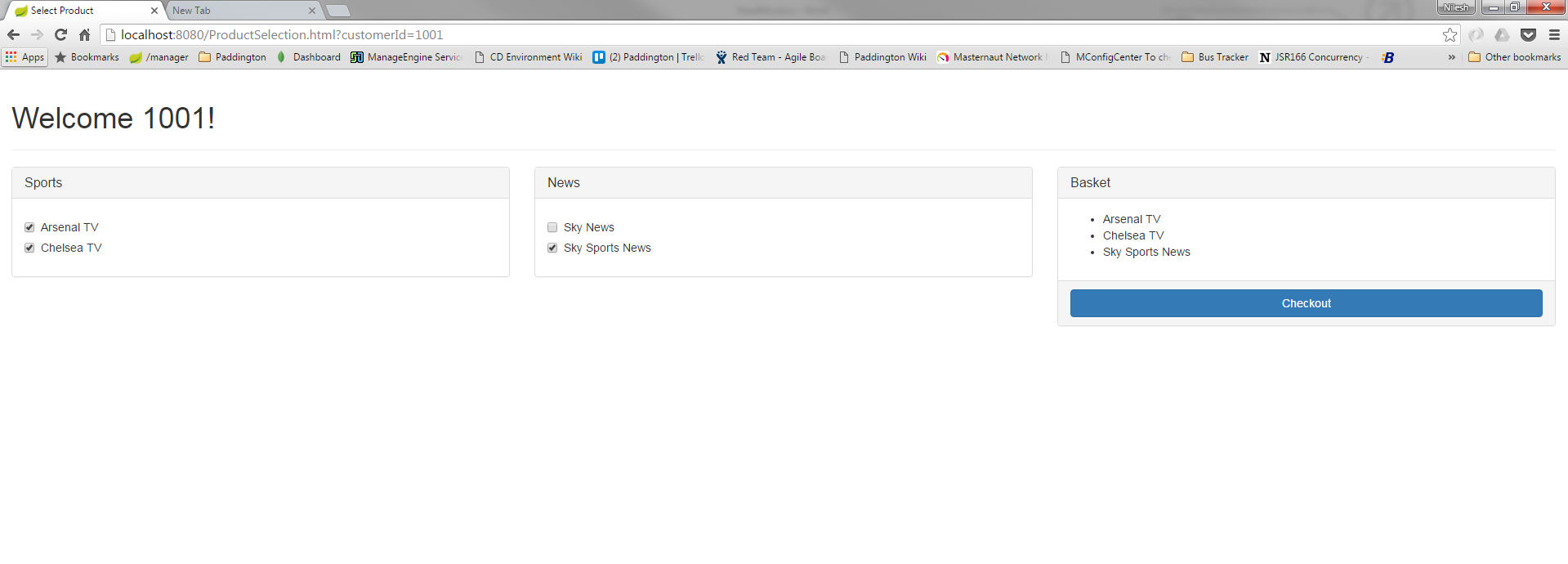
1. The application is up and running now and can be tested for customerId=1001 and customerId=1002.

CustomerId=1001 is set for the locationId=LONDON and

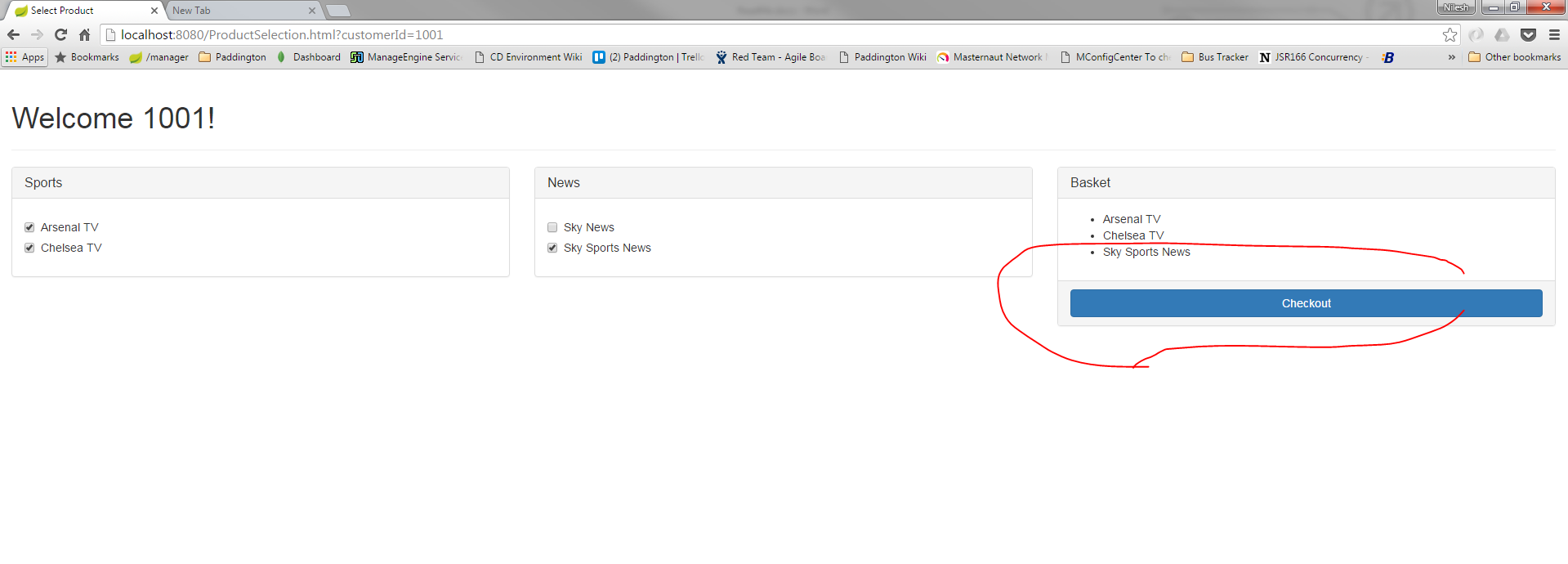
CustomerId=1001 is set for the locationId=LIVERPOOL

See the screenshots below:

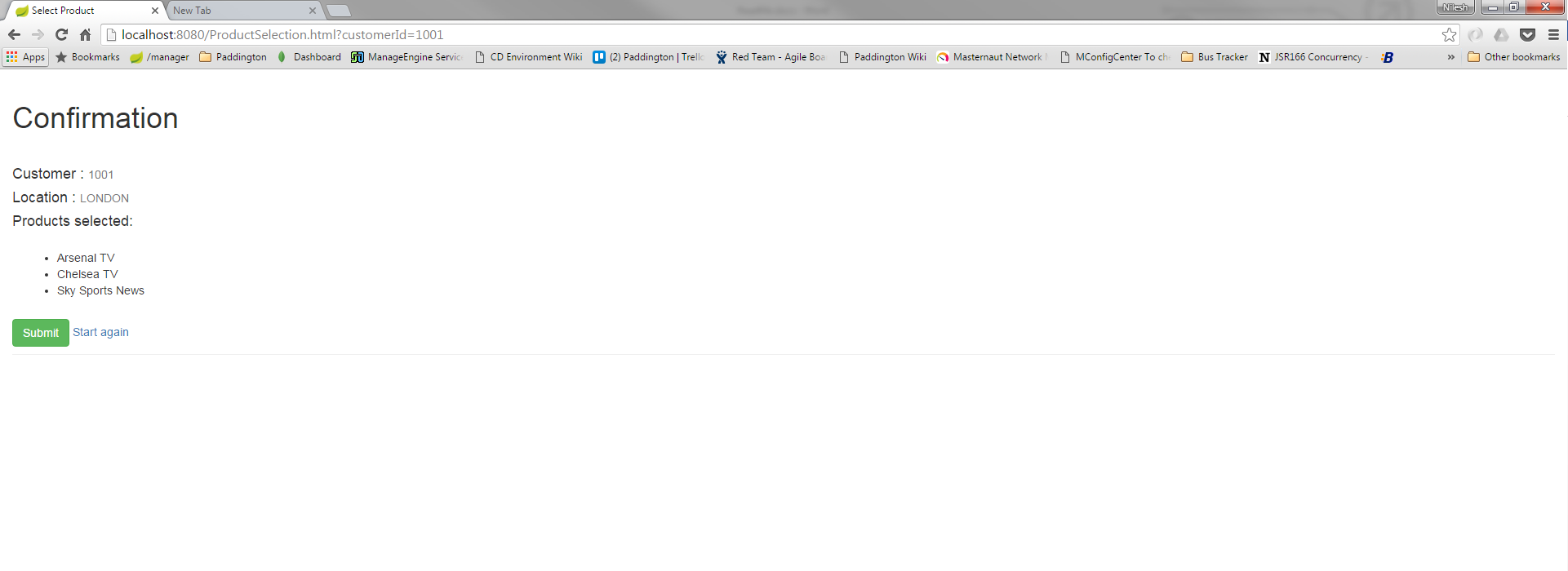
Select Products:



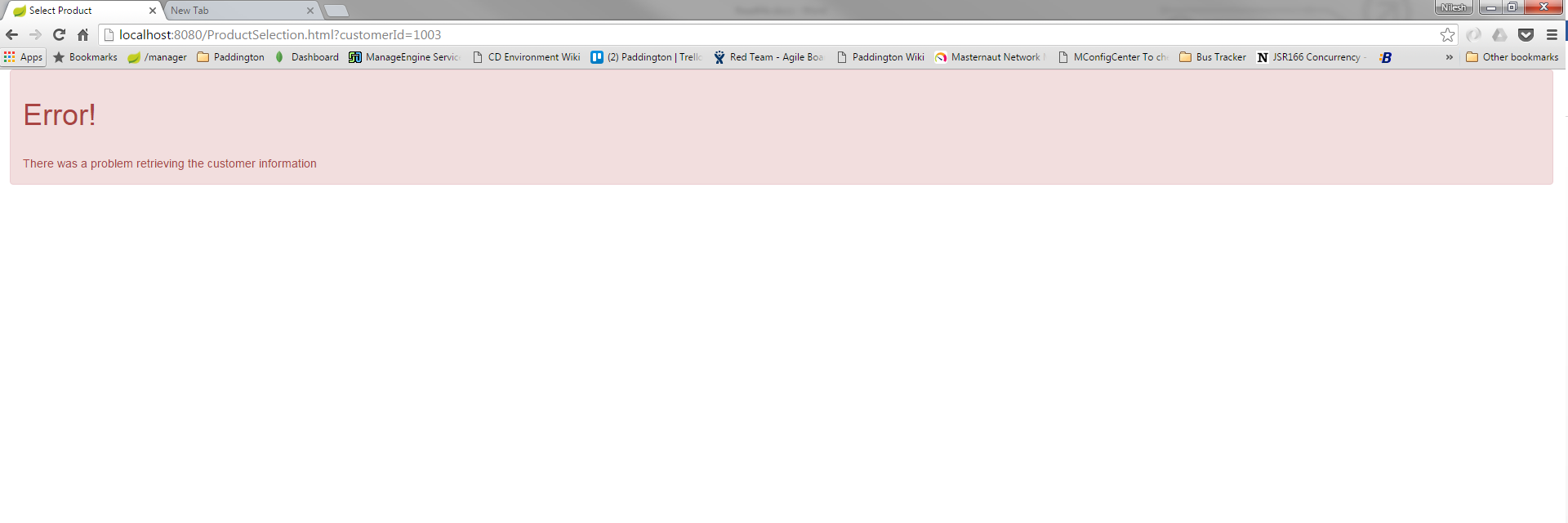
Clicked on checkout:



Confirmation Page:



Any other customerId than 1001 or 1002 will give error as shown below:



Same process can be followed for customerId=1002 using below url in any browser:

<http://localhost:8080/ProductSelection.html?customerId=1002>

1. Any APIs can be individually tested using any REST client:

API URLs are as below:

For CustomerLocationService:

GET <http://localhost:8080/customerLocationService?customerId=1001>

GET <http://localhost:8080/customerLocationService?customerId=1002>

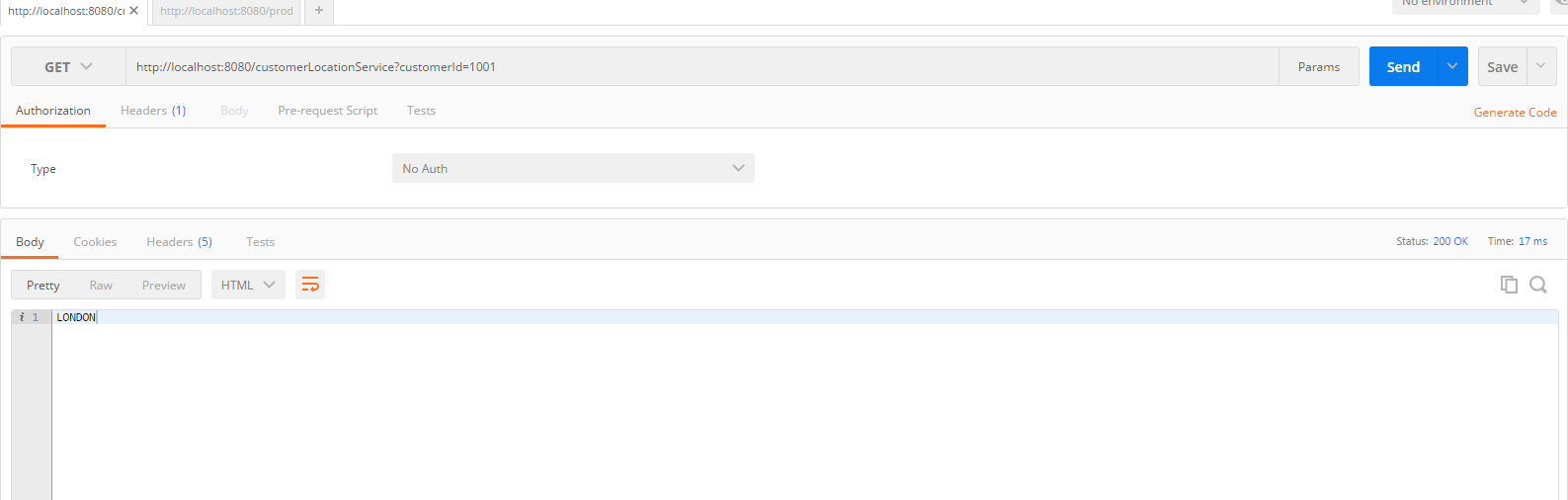
GET <http://localhost:8080/customerLocationService?customerId=1005>

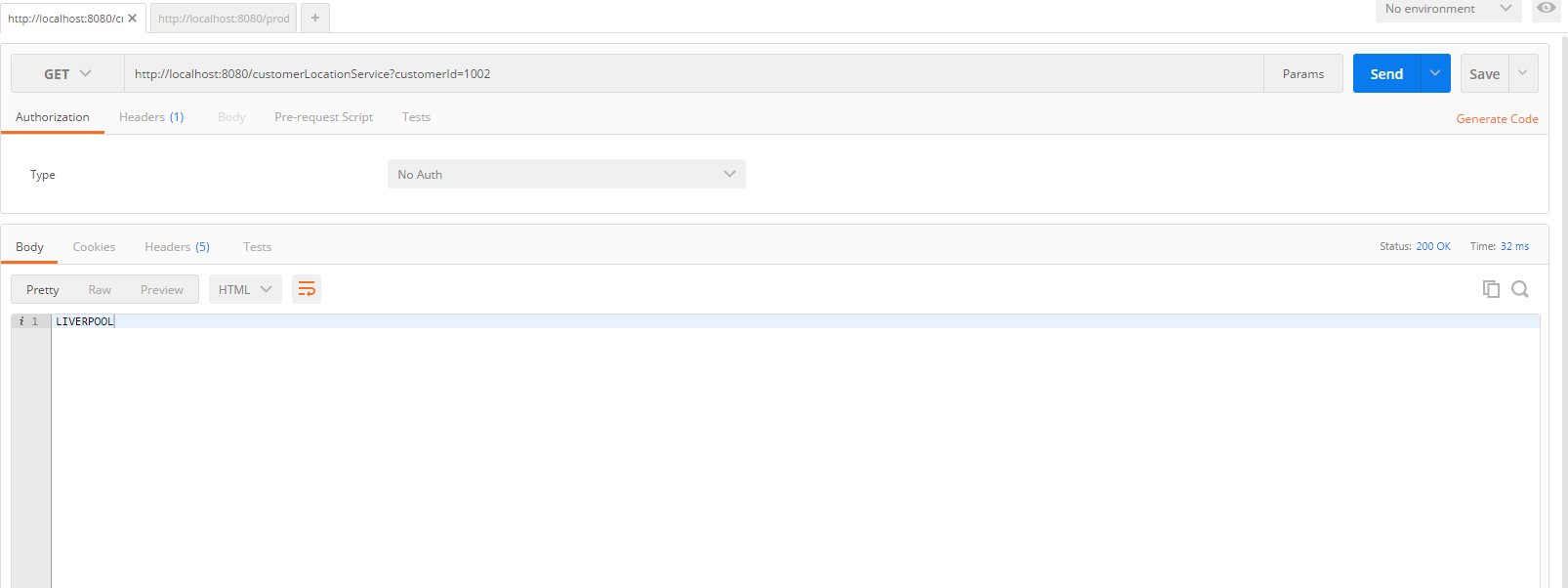
For ProductCatalogueService:

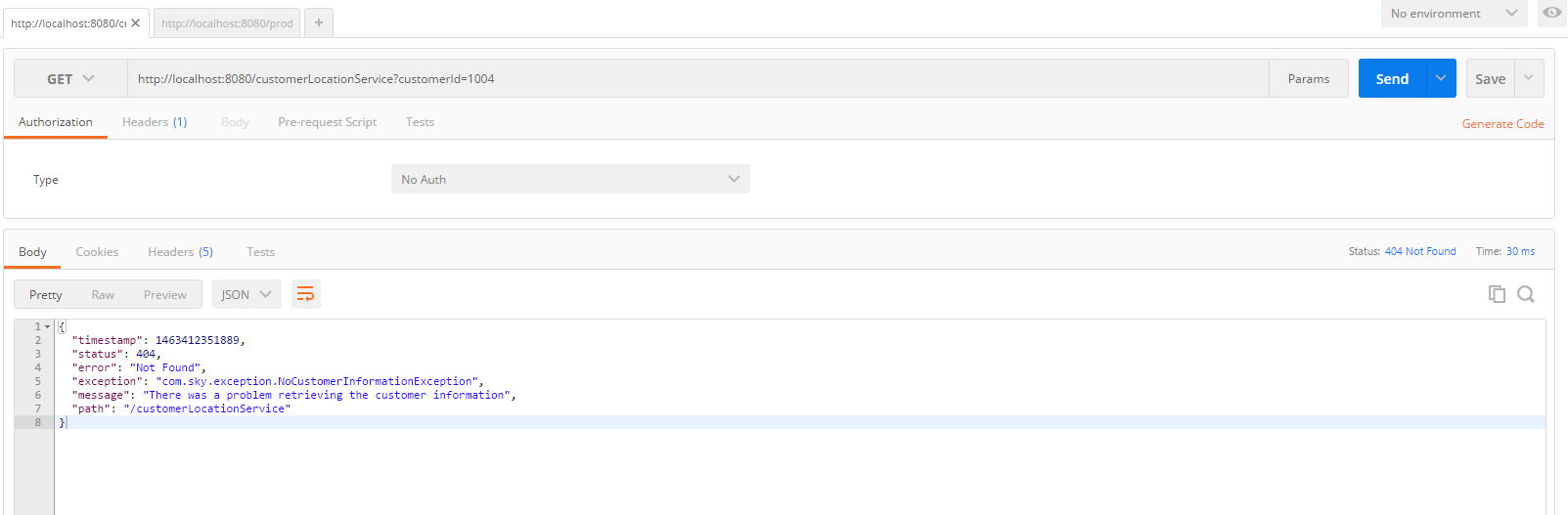
<http://localhost:8080/productCatalogueService?locationId=LONDON>

<http://localhost:8080/productCatalogueService?locationId=LIVERPOOL>

Please refer below screenshots







Note: The project UI might not be fine-tuned though but can be further modified and implemented better way as there can be so many was to implement it.