Order Process System

Design notes:

1. **The primary technologies used are html5/css/ajax/ angular.js, BootStrap framework, google –gson for UI development.**
   spring STS ide(3.7); gradle build tool; Stax parser; Oracle 12c; PL/SQL stored proc using custom object types and oracle collections; Spring MVC(spring 4 version) for backend; multithreading for server side custom caching

2. **UI development**: The Screens are developed using angular /ajax to meet the need of 'Single Page Application'. The pagination, Sorting and fetching json data from the server are implemented in the search screen. The entire SPA is developed using angular.js and both the 'Create Order' and 'Search order' is implemented which is a bit more complex. Create order option is there to upload the data file (Its an xml file!) to the server. Also, informative messages are sent to user screen for success or error scenarios during the search/create order process.
   Order.jsp → angular.js
3. The xml file input is parsed at the server side using Stax. The parsed data is saved as an "Order" object and the same is used to data persistence to the Oracle DATABASE. Data serialization option is also available as an alternative approach to persist data when the Oracle RDBMS is not setup.
4. The data is looked up using JDBC / PLSQL stored procedure call. The json data is handed over to the client upon successful lookup. The browser parses the data and displays in table. The pagination feature is incorporated so that large data is split into multiple pages and user is given a handle to browse the pages. The sorting feature is also added enabling the user to sort entire data set in ascending or descending pattern and pages are displayed in sync to pagination design.
5. Caching is implemented at server side for fast accessing the recently searched orders and this improves application performance while offering the fast response to the users. Implemented multithreading option.

6. **My thoughts**: current project is a model to demonstrate only the working features of the application. The following features can still be added for making the application more responsive and efficient.
   1) EH cache can be implemented which is robust.
   Screen look and feel: The Order jsp page can be further improved for look and feel.
   The secure login feature can also be added where we can address the users based on two groups; admin and common users (development is underway by letting the admins to update the order!).

7. **How to run the application:**
   1) Two options to save the data; (a) without Oracle RDBMS system (serialize the data object to the hard disk on server side). Or (b) save it into Oracle database.

      I (a). **Serialize the data object to disk:** since 'programing to interface' pattern is followed, it is very simple to switch between 1(a) and 1(b). The xml data is initialized as 'order' object instance. This object is serialized and written to disk. Although this is

functional but less responsive unless to test the UI/front end development. Also this feature is useful to test the flow when the Oracle RDBMS is not accessible.

The buz logic(search and save order) is implemented in '`OrderSerializationHelper.java`'.
`@Repository("orderDao"):` Uncomment this annotation enabling bean creation for the above java file.
At the same time comment out the same line in all files, '`OrderDBHelper*.java`'

II (i). **Oracle RDBMS:** Swap to Oracle mode by reversing as mentioned in the above step, I. That is, uncomment `@Repository("orderDao")` in one of the files '`OrderDBHelper*.java`' and comment out the rest of the java files.

Prepare database: (a) Create the database tables, custom objects, and Sequence and (c) Create the oracle stored procedures

ii) import the project file into your workspace. Prepare the project to be eclipse loaded if using java sources (run command, `gradle eclipse`). Run, **"`gradle jettyRun`"** to start the jetty9 (lightweight) server. Also there is an option provided in build script to start the apache server. It is a personal preference which server can be used for deployment. When the server is up, launch the web browser, and enter the url: http://localhost:9999/venu/order.do (this is as per my config. It may slightly change depending on the respective machine configuration)

iii) oracle db properties: change accordingly in the file, 'application.properties'

```
user=venu
pass word=Summer2015
url="jdbc:oracle:thin:@localhost:1521:LOGISTICS"
```

8. **i) Sample xml data file used for loading into the system:**

```
<?xml version="1.0" encoding="utf-8"?>
<order>
  <from zip="80817" state="CO" city="COLORADAO SPRINGS"/>
  <to zip="96821" state="HI" city="Honolulu"/>
  <lines>
    <line weight="10000.1" volume="14" hazard="false" product="Engine Block"/>
    <line weight="200.55" volume="8" hazard="true" product="cable"/>
    <line weight="100.1" volume="14" hazard="false" product="plugs"/>
    <line weight="165" volume="8" hazard="false" product="electronic controls"/>
    <line weight="1008.1" volume="14" hazard="false" product="Engine Block"/>
    <line weight="30.55" volume="8" hazard="true" product="Liquid Nitrogen"/>
  </lines>
  <instructions>Transport in secure container</instructions>
</order>
```

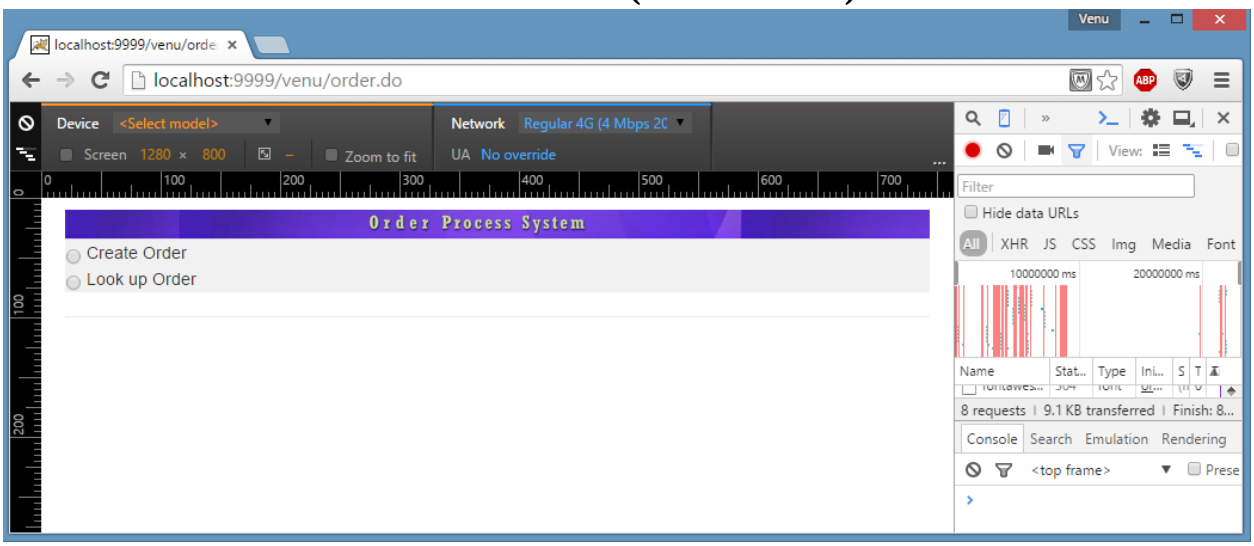
ii) Schema(order.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  elementFormDefault="qualified">
  <xs:complexType name="order">
    <xs:sequence>
      <xs:element name="from" type="location"/>
      <xs:element name="to" type="location"/>
      <xs:element name="lines" type="lines"/>
      <xs:element name="instructions" type="xs:string"/>
    </xs:sequence>
```

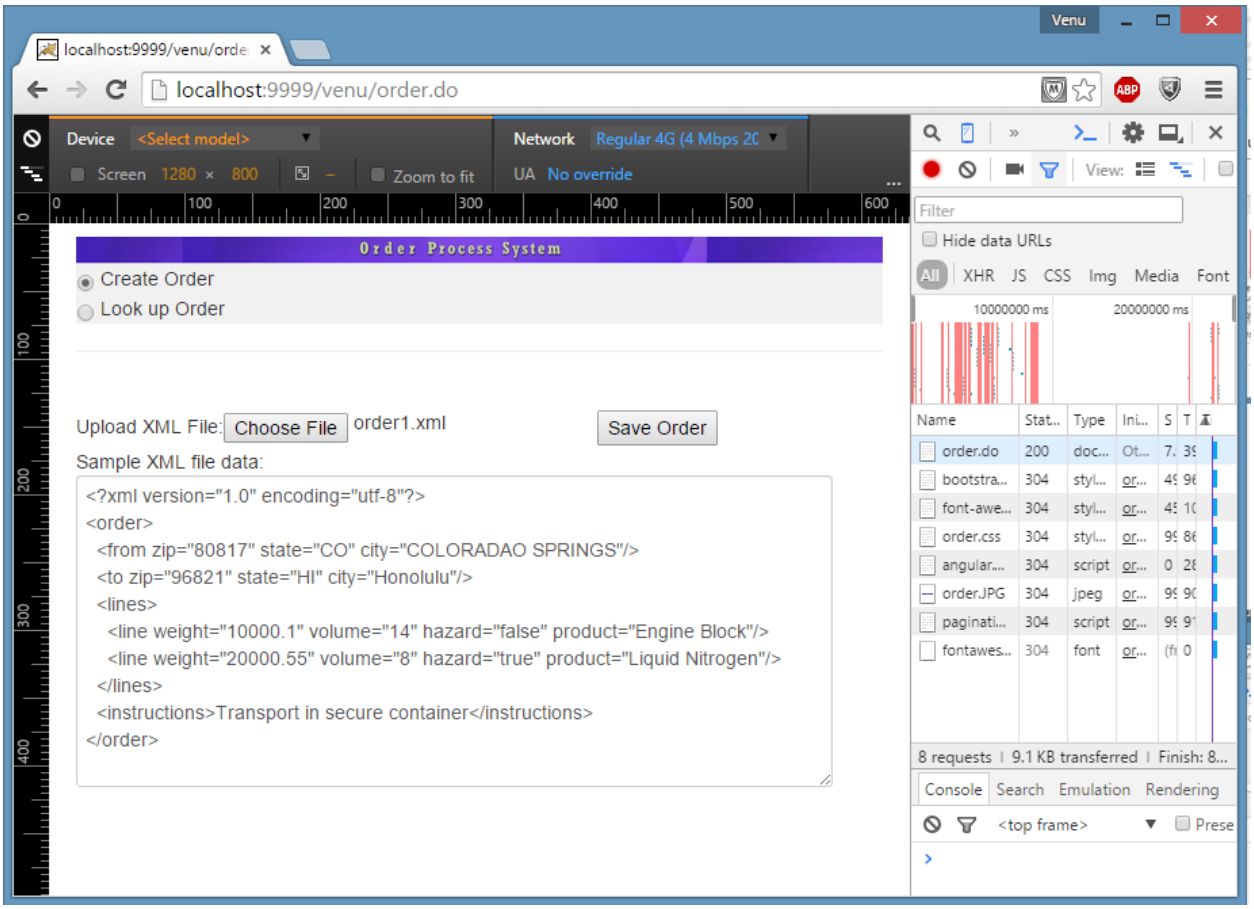
```
    </xs:complexType>
    <xs:complexType name="location">
      <xs:attribute name="city" use="optional" type="xs:string"/>
      <xs:attribute name="state" use="optional" type="xs:string"/>
      <xs:attribute name="zip" use="required" type="xs:string"/>
    </xs:complexType>
    <xs:complexType name="lines">
      <xs:sequence>
        <xs:element name="line" maxOccurs="unbounded" type="line"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="line">
      <xs:attribute name="hazard" use="optional" default="false" type="xs:boolean"/>
      <xs:attribute name="product" use="required" type="xs:string"/>
      <xs:attribute name="volume" use="required" type="xs:double"/>
      <xs:attribute name="weight" use="required" type="xs:double"/>
    </xs:complexType>
    <xs:element name="order" type="order"/>
</xs:schema>
```

9. Navigation and application flow:

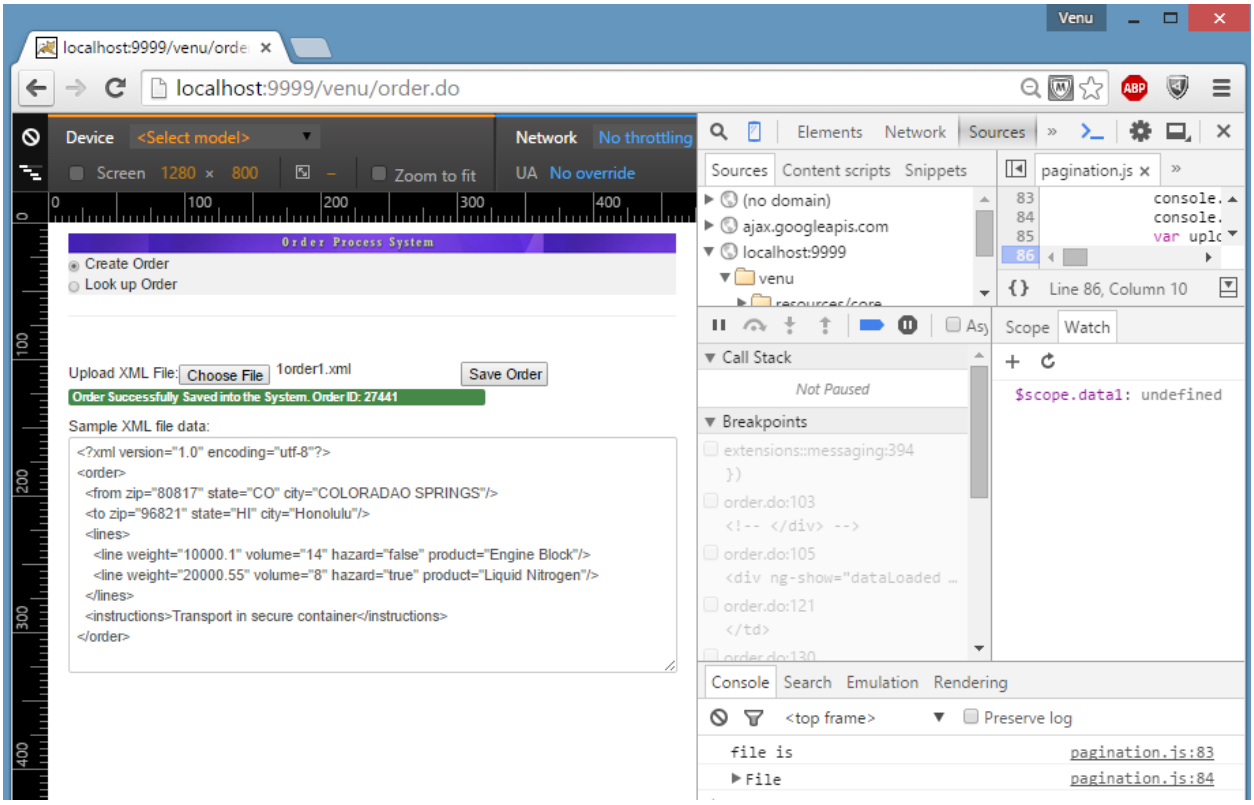
Screen1 (initial screen)



Screen:2 (Uploading xml file; ‘order’ data)

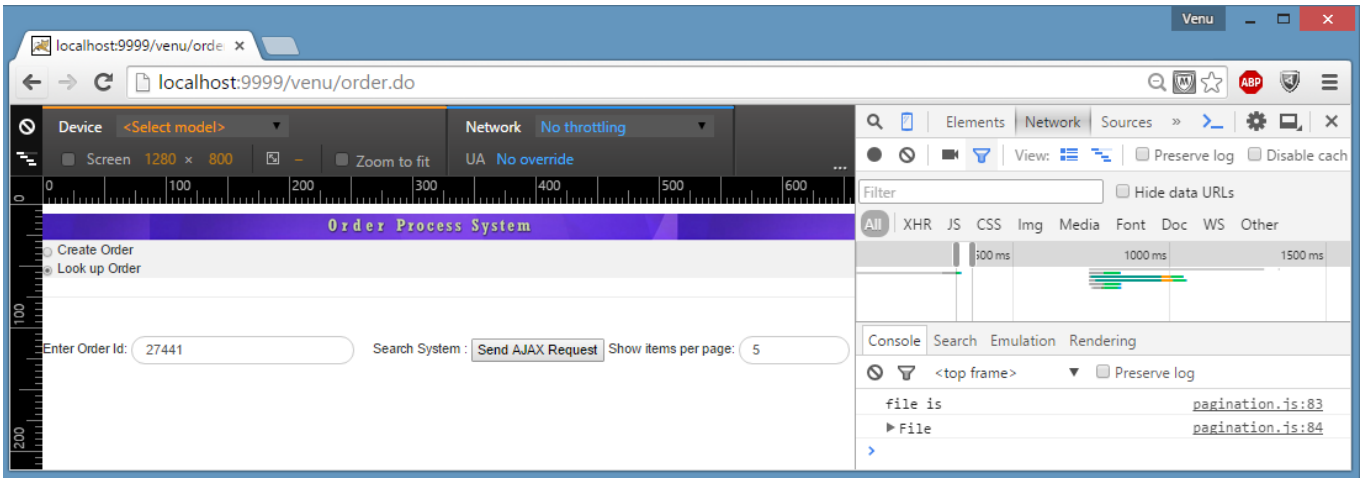


Screen: 3 (File Upload completed, saved to DB)



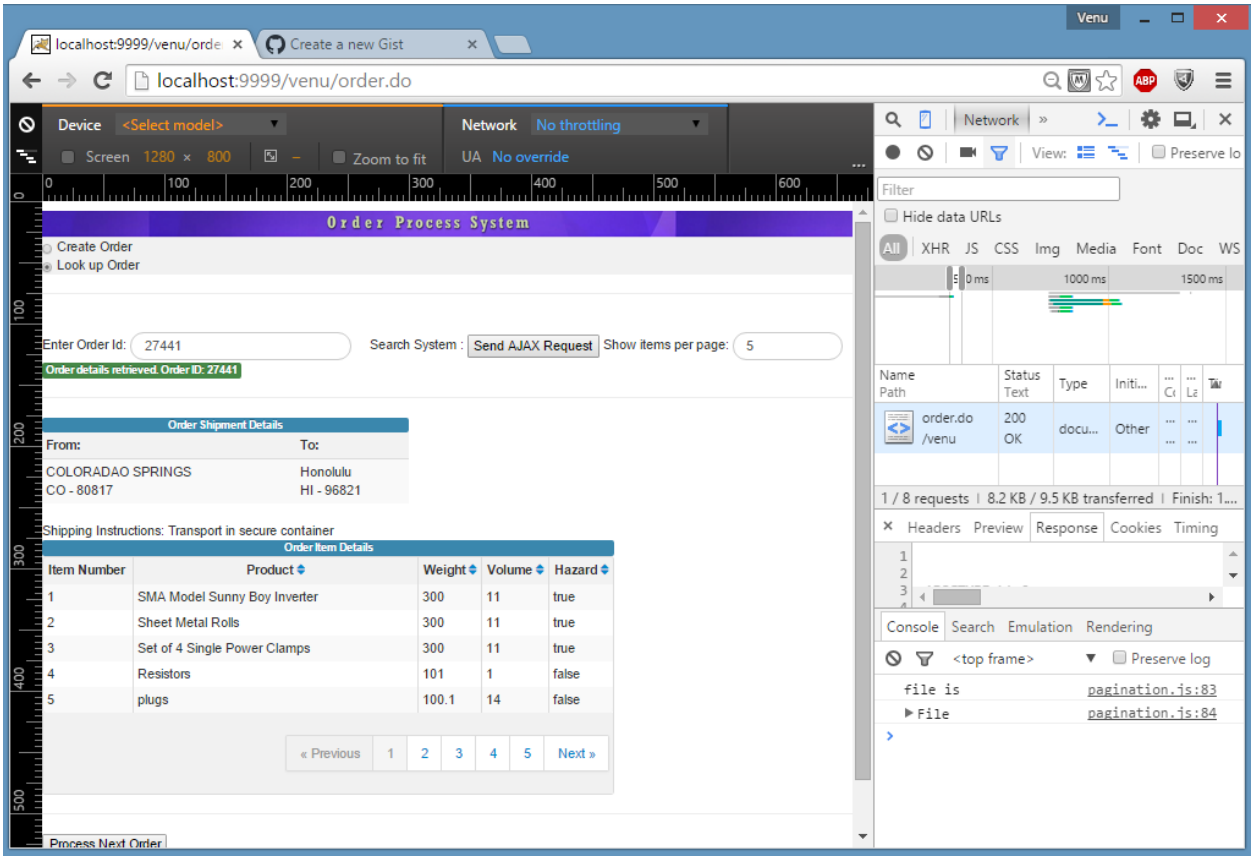
Screen: 4.i. (search order id: initial screen)

Initial search screen loaded. Enter Order ID.



Screen: 4.ii. (results of search: Tabular display)

Results are displayed in a table structure. Data in the columns are sortable (toggle reverse option). Default page size is ‘5’ and pagination feature is also incorporated.



DATAMODEL: Screen: 6