Design notes:

1. **The primary technologies that I have used are jquery/css/ajax google's angular.js, google – gson for UI development.**
   BootStrap framework, spring STS ide(3.7); gradle build tool; Stax parser; Oracle 12c; PL/SQL;; Spring MVC(spring 4 version) with no xml config files but rich set of Annotations; jetty9 and apache-tomcat-7.0.63 server servlet3 container and some other open source frame works like logback,

2. UI development: The Screens are developed using ajax to meet the need of 'no screen refresh'.

   The work is currently going on for sorting the order items in the search screen using angular.js lib.
3. The xml file input is parsed at the server side using Stax. There are many choices available like DOM/SAX/JAXB etc. but Stax is chosen for less in memory foot print which is a push based event driven system. Java sources are found in the attached zip, java soures.zip. (war file, java source files, plus the thread pool dev work)
4. The parsed data is saved as an "Order" object and the same is used to data persistence to the Oracle DATABASE at the service layer.
5. The data is persisted using JDBC as needed by the Assignment requirement. For the data insertion, I used one single PL/SQL stored procedure call. Multiple network calls cane be avoided which makes the application more responsive. The PL/SQL is very efficient/fast. The stored procedure is called only once for a given request.
6. For search order task, Data Look up is done using one single sql statement. The returned data is used to make the Order instance. I used this to convert to json data and handed over to the client side. The json object is parsed to show the data as collapsible/expandable list (tree list).
   This is accomplished by just jquery /css. No third party libraries used.
7. Cache is implemented on the server side to maintain the latest order object be available in the system. This avoids unnecessary round trip network calls to the oracle database. Refer the LRUMap as introduced by 'Apache' which is a special type of map that can be implemented in cache systems. For LRUCache we can mention the max map elements. The size selected should be kept optimal as this memory becomes overhead on the hard ware resources if its too big. The thread (set as daemon thread) will run once for the time mentioned in the 'timerInterval'; and identifies if a particular map element (cache obj) is over living by the time specified as mentioned by 'timeToLive' and deletes those map elements from the cache. Max elements are the allowed max cache size at any given point of time.

```
 * @param timeToLive --> 36000 sec
 * @param timerInterval --> 180 sec
 * @param maxItems  --> 10
```

1. **My thoughts**: My 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) Cache can be implemented to the system by using one of the open source frame works like, 'hibernate' or other ORM frame works.

2). Connection DB pooling (like, apache DBCP) can be added to the existing project to enhance performance. The ThreadPool plus, the DBCP is a real industry standard pattern.

3). Screen look and feel: The Order jsp page can be further improved for look and feel. Also, user can be given one more option of entering the order details in the text input filed in addition to the currently implemented feature of uploading the file. Spring Boot can also be used to develop the java application for rapid application development.

4). In a real industry standard, I can think of a large number of request hits to the application. Based on this assumption, the further guaranteed way of implementing failed/missed requests, if any, can be better addressed by JMS. I have addressed similar feature in my previous projects.

2. **How to run the application:**
   1) **Prepare database.**
      Create the database tables
      Create the Sequence
      Create the oracle stored procedure
   2) **import the project** file into your workspace. Prepare the project to eclipse loaded if using java sources (run command, gradle eclipse). Run, **"gradle jettyRun"**. 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 your configuration)
   3) **oracle db properties:** user=venu
      pass word=Summer2015
      url="jdbc:oracle:thin:@localhost:1521:LOGISTICS"

3. Below are the screen shots
4. Sample xml file for upload: order.xml

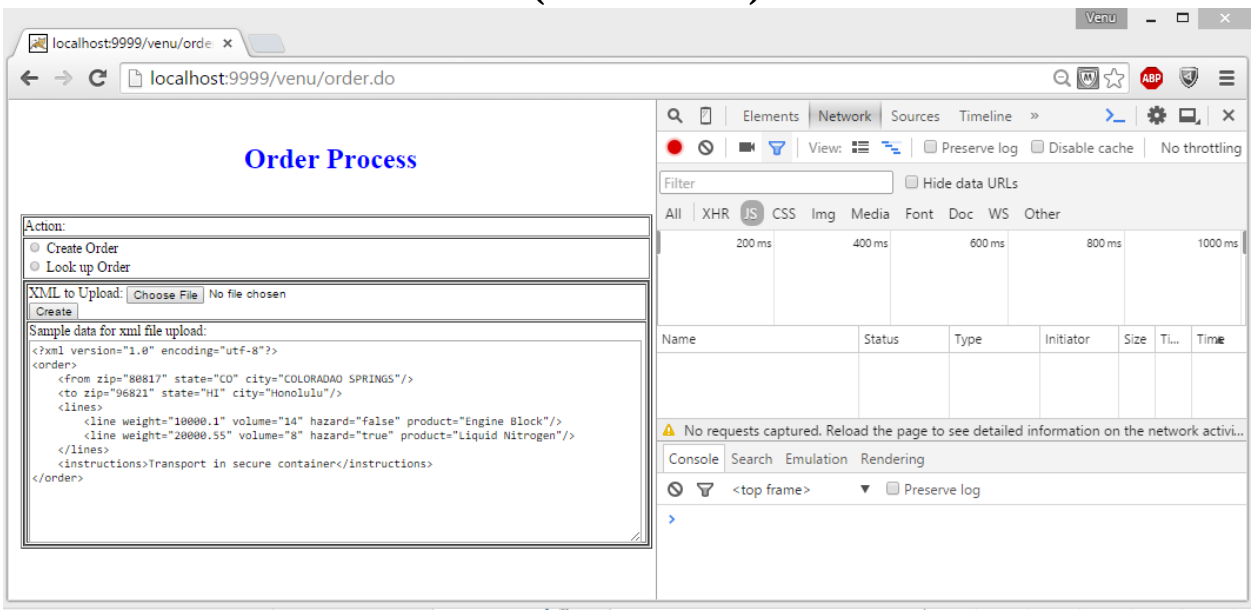
```
<?xml version="1.0" encoding="utf-8"?>
<order>
   <from zip="80817" state="CO" city="COLORADAO SPRINGS"/>
   <to zip="96821" state="HI" city="Honolulu"/>
   <lines>
      <line weight="10000.1" volume="14" hazard="false" product="Engine Block"/>
      <line weight="200.55" volume="8" hazard="true" 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>
```

19. Schema(order.xsd)

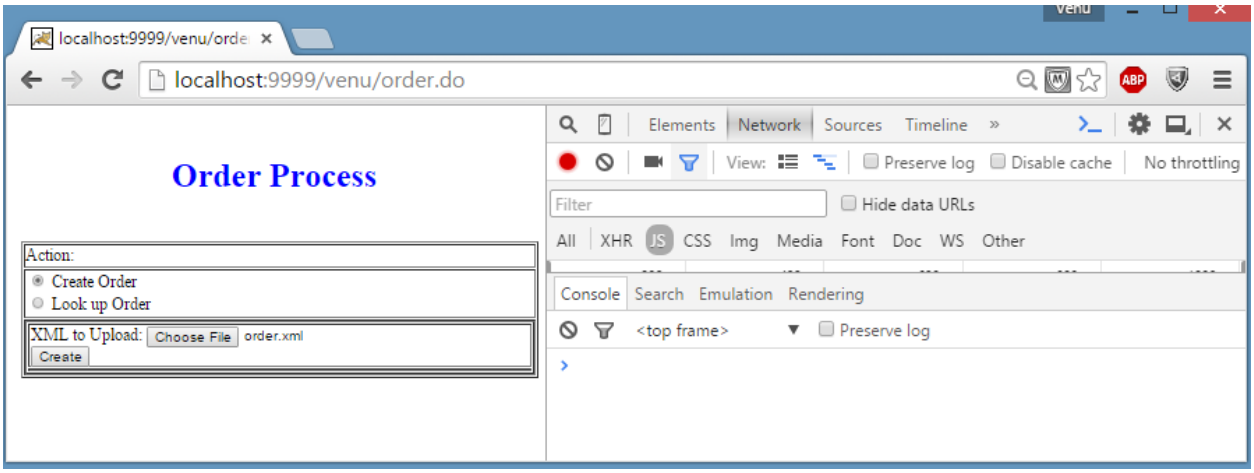
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
   <xs:complexType name="order">
      <xs:sequence>
         <xs:element name="from" type="location"/>
         <xs:element name="to" type="location"/>
         <xs:element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

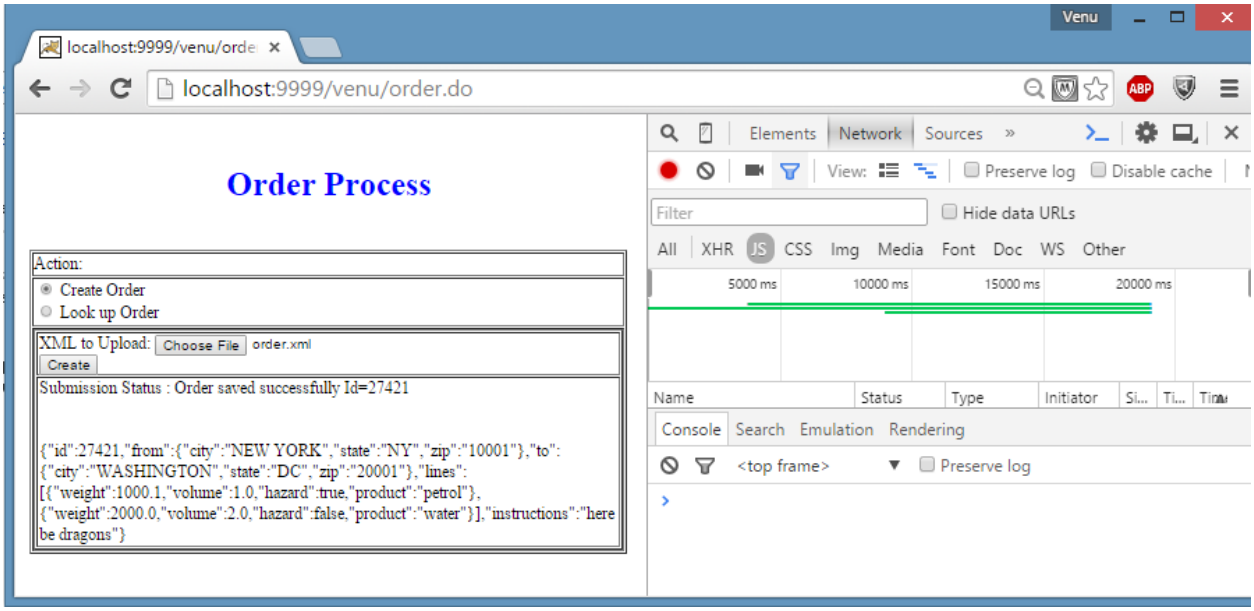
Screen1 (initial screen)



Screen:2 (Uploading order file)

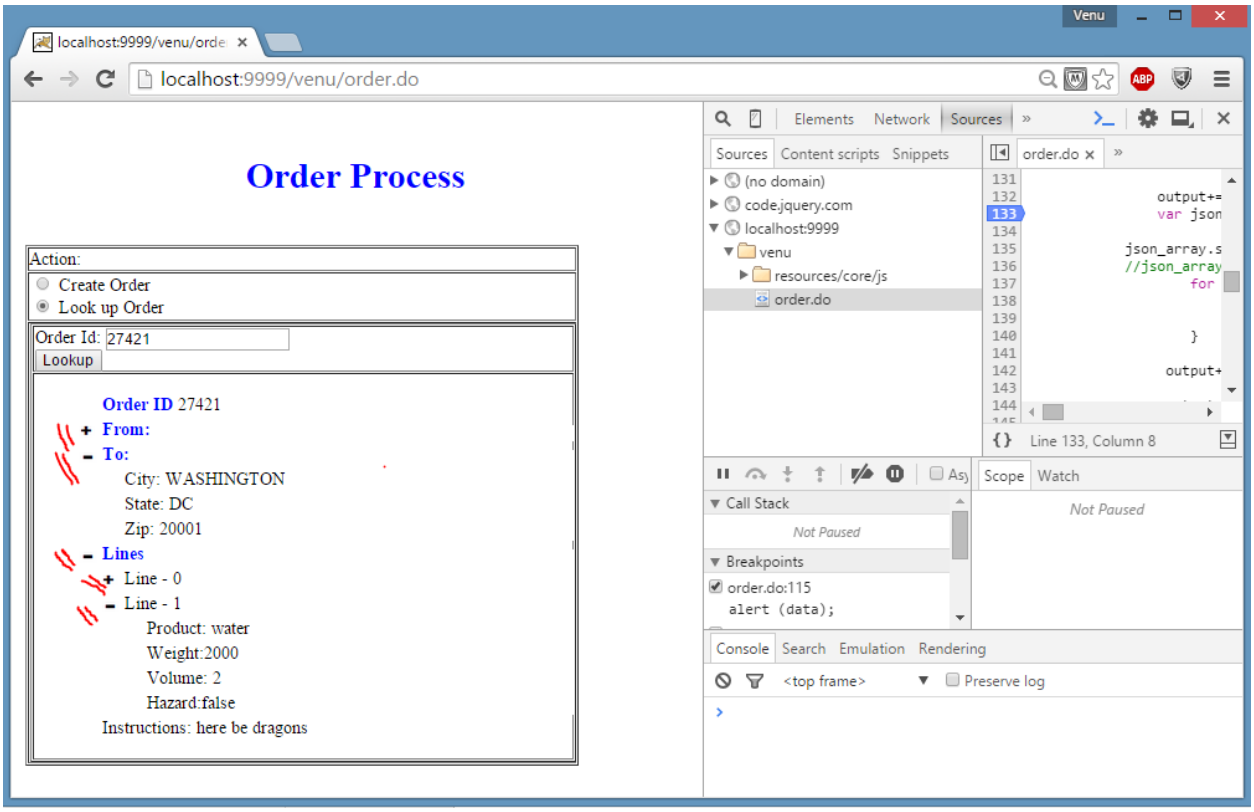


Screen:3 (File Upload completed, saved to DB)



Screen:4 (results of search : Tree structure)

Results are displayed in a tree structure which are collapsible/expandable. Red markers are some of those.



DATAMODEL: Screen: 6