Workshop 1: A Request Based Approach, the Java Servlet API

Objectives

The goal for this workshop is to expose a given OO-model on the web (a web shop). We'll create a request based application to interact with a part of the model (the ProductCatalogue). You need the following tools and skills;

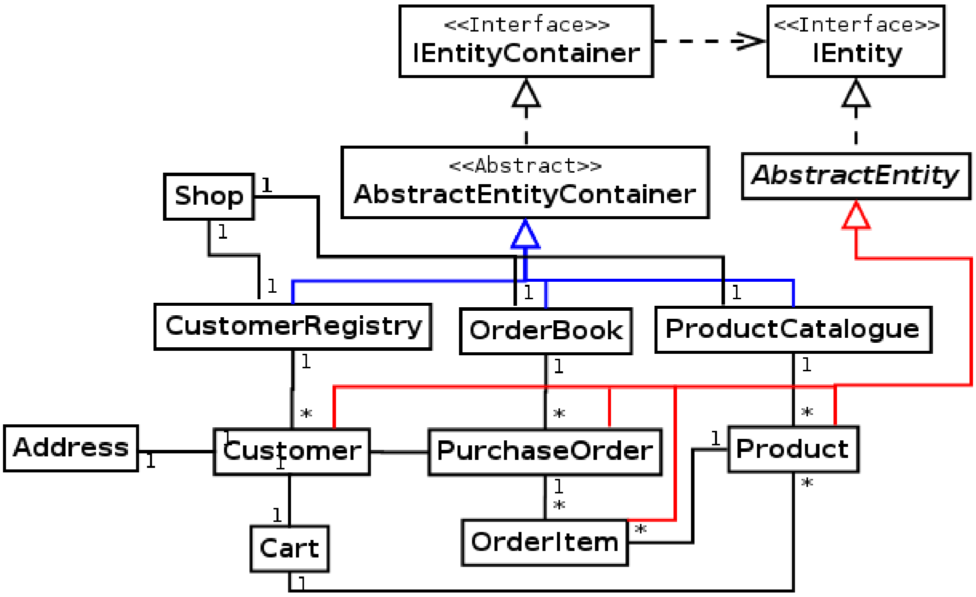
- Environment: NetBeans IDE (including Maven, Tomcat Server) or similar.
- Basic XML, HTML and CSS and Bootstrap.
- JEE Web applications and the Servlet API (Servlets, Java Server Pages (JSP), JSP Standard Template Library (JSTL).
- The FrontController JEE design pattern.
- The Post-Get-Request (PRG) pattern.

Please: Inspect code samples from the lectures (on course page)! Everything you need should be there. Will hopefully save you a lot of time!

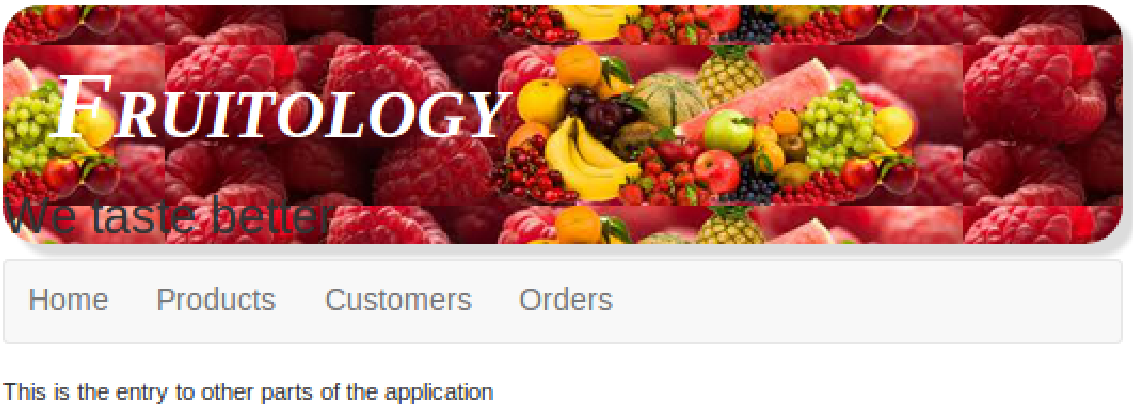
Final date : See Course Page

1 The shop model

We will use a basic model of a web shop during the workshops. Below is an UML class diagram of the model;



1. Download the model from course page > Workshops. Unzip and open project in NetBeans (it's a Maven Java standalone application).
2. Inspect project. Check classes: Shop (note some default data), ProductCatalogue, OrderBook and CustomerRegistry.
3. Switch of testing for ordinary builds: Tools > Options > Java > Maven, check "Skip Tests for any...."
4. Build the project. Will install the model into local Maven repo (~/.m2 directory) as shop-1.0-SNAPSHOT.jar. Try to find it. When installed in repo it's possible for other applications to add a Maven-dependency on the model (we will use the same model in all workshops).
5. There are a few tests. Run the tests.
6. Inspect shop-1.0-SNAPSHOT.jar in the Files view (tab). Important to check that everything really is included in the jar (more important later).



This is the entry to other parts of the application

Figure 1: Application main page (index.jspx)

2 Design

Now we will wrap the model in a request based web application. The task for you is to implement the Products part of the application (CRUD operations for the Product catalogue).

Note The final project development structure is in Appendix

Figure below shows the overall design. Assume the arrow represents the path of the request object:

- If authentication is implemented, see below, the request hits the AuthFilter (not present right now).
- Depending on URL, request enters Shop- or Productservlet. ProductServlet interacts with ShopModel to read or write product data then navigates. ShopServlet only navigates between different parts of the application (Products, Orders, ...). Any data needed in the pages are set in the servlets (using request.setAttribute or session.setAttribute).
- Servlets forward or redirects to template.jspx which in turn includes needed content using <jsp:include..>.
- Pages access data set in servlets using EL-expression (like ${...}).
- The Listener is used to put a reference to the Shop model into the ApplicationScope at application start up (model will exist as long as application runs).

The graphical design uses the Bootstrap CSS library.

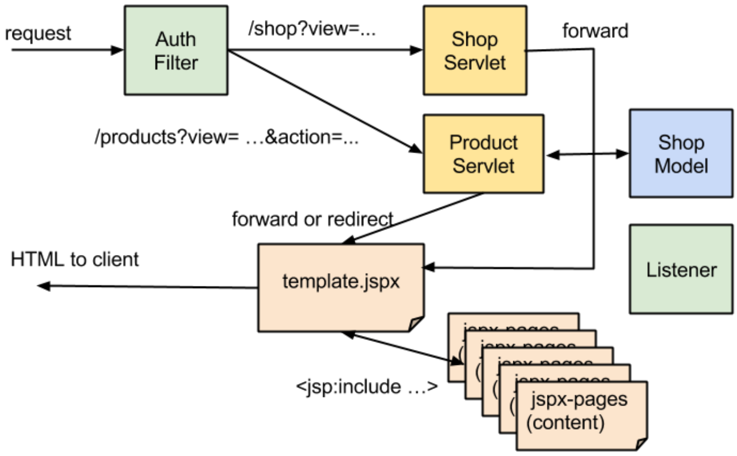


Figure 2: Design for request based approach.

3 Preliminaries

1. Download an application skeleton from course page > Workshops. Unzip and open project in NetBeans (it's a Maven Java Web application). There are possible warnings or errors, ignore for now.
2. There should be a dependency on the shop model. Inspect pom.xml.
3. Build the project (Maven possibly will download a lot, be patient). Inspect generated .jar file in Files view. Now all warnings and errors should be gone, if not contact assistant.

   Note We'll use Java 1.7 (Java 7) for our projects. Mark project, right click > Properties > Sources (should be Java 1.7) and Compile (Java 1.7 or Java 1.8).
4. Familiarise with the Web application structure.
5. Inspect Services-tab > Servers in NetBeans. You should find an Apache Tomcat server. Right click icon > Properties, inspect.

   Note If "Enable HTTP Monitor" is checked it's possible to inspect incoming HTTP requests in HTTP Server Monitor window (pops up at run). Very useful.
6. Start Tomcat, right click > Start. Use a browser to visit http://localhost:8084 (default for Tomcat admin pages). Stop Tomcat.

   Note There are always two administrative applications running in Tomcat, shown as / and /manager. Don't touch!
7. Mark project > Properties > Run > Select Tomcat (possible already selected).
8. Mark project > Run. Tomcat should start (log windows opens in NetBeans) and application welcome page (see web.xml) should show up in default browser (adjust browser Tools > Options > General > Web Browser).
9. Try to access different JSP's using the browser address field. Conclusions?
10. Compare browser address field with content in file Web Pages/META-INF/context.xml. Change path in context.xml and run again. As expected? Reset!

    Tip To speed up the deployment of the project, specially when testing small changes to server side code, use "deploy on save", NetBeans will compile and deploy the application on every save (at severe exceptions possible have to Build/Run again). Recommended!

    In an existing application. Mark application, right click > Select Properties > Run > check Deploy on Save.

    Select Properties > Build > Compile > Compile On Save: For both application and test...

Tip Indexing of local Maven repo is very time consuming. Select Tools > Options > Java >Maven > Index Update Frequency: Never

Warning Use clean and build generously. NetBeans seems to cache too hard ... i.e. unclean builds.

11. It's possible to debug a Web application but it's somewhat heavy. Use of Logger is recommended to trace the execution. Point into some editor window, right click > Insert code ... > Logger .... Use like this (in general let NetBeans generate as much code as possible in particular constructors/setters/getters):

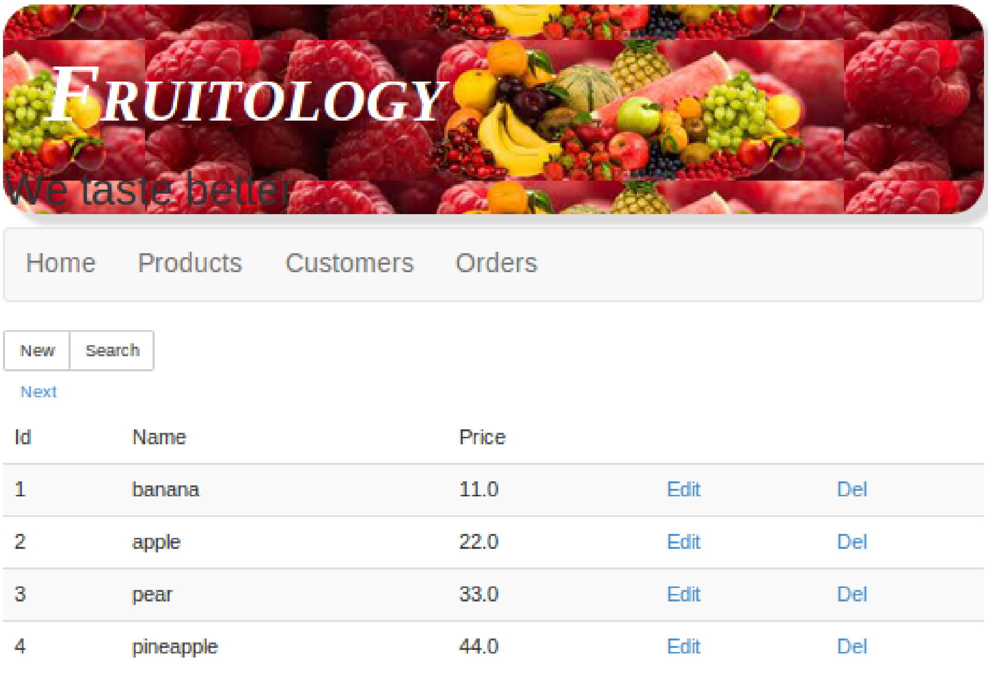
```
LOG.log(Level.INFO, ".. some message... {0}", somevalue);
```

4 Implementation

You should only implement the ProductCatalogue functionality

1. Files to work on: the files in the /WEB-INF/jsp/products folder, the ProductServlet. If implementing authorization add a filter.

2. The first goal is to display a web page with a table of products like below. This page should be reachable from "Products" in the main menu. The table is located in products.jspx. Start out with a simplified table with no navigation and no links. Work with the JSP and the ProductServlet. Use JSTL and EL to generate the table (a loop).

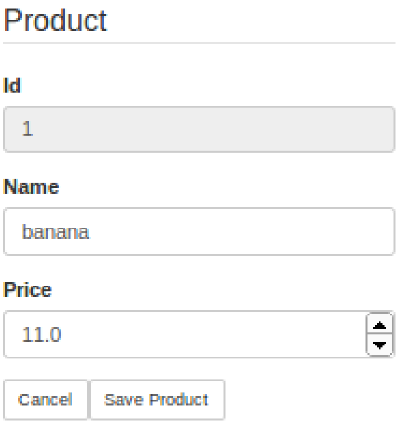


Id	Name	Price		
1	banana	11.0	Edit	Del
2	apple	22.0	Edit	Del
3	pear	33.0	Edit	Del
4	pineapple	44.0	Edit	Del

3. Add navigation to products.jspx using JSTL <c:if ...> and EL (or other, feel free to test). Inspect ShopServlet for some ideas.

4. Master detail Implementation

    - Add links to to the table pointing at editProduct.jspx and delProduct.jspx. Use EL to add request parameters for identification of object to edit or delete (NOTE: Id never changes).
    - Add a link to the addProduct.jspx to make it possible to add new products.
    - All details pages are composed of a form with a few controls like below.



Product

Id

1

Name

banana

Price

11.0

5 The Post Redirect Get (PRG) Pattern

Check that the PRG pattern is correctly implemented.

6 Authentication and Authorization

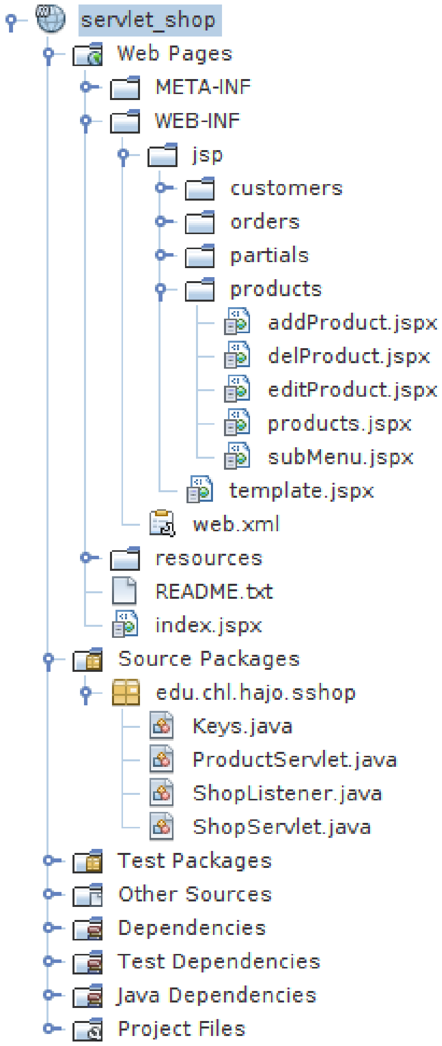
(Optional) As a means of auth-entication/orization we'll use a Filter. The general idea is;

- All requests to URL "/products/*" will hit the filter.
- Filter checks if there's a user-object in the HttpSession-object (if session exists).
- If so the request is passed through.
- Else there's a forward to some login page. Login (and logout) is handled by an AuthServlet. If login succeeds the Servlet puts the user-object into the HttpSession. Else an error message is displayed in the login page.

Create a User, an AuthFilter, an AuthServlet (in package auth) and a public jspx-page (login.jspx).

APPENDIX

Final Project development structure.



- servlet_shop
  - Web Pages
    - META-INF
    - WEB-INF
      - jsp
        - customers
        - orders
        - partials
        - products
          - addProduct.jspx
          - delProduct.jspx
          - editProduct.jspx
          - products.jspx
          - subMenu.jspx
        - template.jspx
      - web.xml
    - resources
    - README.txt
    - index.jspx
  - Source Packages
    - edu.chl.hajo.sshop
      - Keys.java
      - ProductServlet.java
      - ShopListener.java
      - ShopServlet.java
  - Test Packages
  - Other Sources
  - Dependencies
  - Test Dependencies
  - Java Dependencies
  - Project Files