Workshop 3: A Component Based Approach, Java Server Faces

Objectives

Same as previous workshops (expose the ProductCatalogue). We need;

- GlassFish application server.
- Java Server Faces (JSF) and Facelets.
- Context and Dependency Injection (CDI).
- Bean validation.
- PrimeFaces, a higher level component library for JSF.

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 JSF request cycle

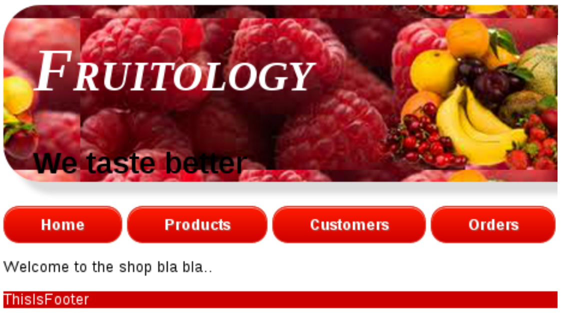
Recommended is GlassFish.

1. Download the JSF request cycle demo from course page. Run and try to get an understanding. What happens? When?

Note The URI's, to trigger JSF, see <servlet-mapping> in web.xml (similar to /rs/ for REST).

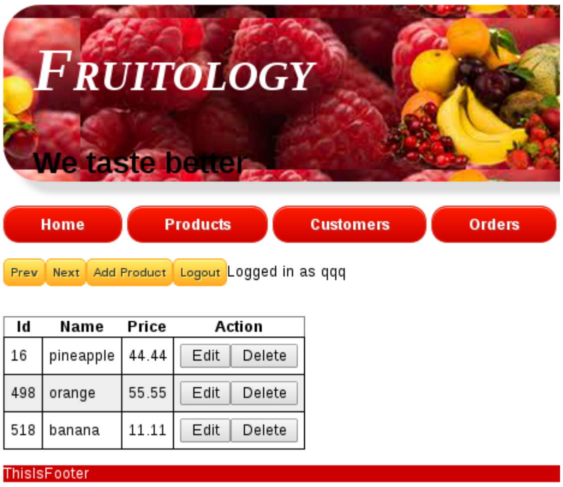
2 A JSF front-end to the Model

Yet another way of exposing the shop. Front-end looks very similar and the projects final structure is as usual in Appendix.



Welcome to the shop bla bla..

ThisIsFooter



Logged in as qqq

Id	Name	Price	Action
16	pineapple	44.44	Edit Delete
498	orange	55.55	Edit Delete
518	banana	11.11	Edit Delete

ThisIsFooter

2.1 Creating the overall layout

We'll use Facelets to create the overall page layout (a template).

Note Facelets uses XHTML, only.

1. Download the skeleton app from course page and open in NetBeans. Inspect, note new xml config-files in WEB-INF!
2. It should be possible to run and add a product. Try!
3. Edit and delete pages are missing. So ...
    a) Inspect /WEB-INF/template.xhtml, the Facelets template file.
    b) Create Facelets pages for edit and delete using the template, see addProduct.xhtml, see Appendix.
    c) Create backing beans for edit and delete pages. Both should extend ConversationalBase and use the conversational scope (to be able to remember which id to edit or delete). Don't connect to the shop yet, see 2.2.
    d) Create the buttons for edit and delete in table on products.page (see above).
    e) Connect pages, backing beans and buttons.
4. Add navigation. When adding, editing or deleting is finish we should return to products.xhtml. See faces-config.xml.
5. Make the front-end work (possible use System.out to trace).

2.2 CDI injection

Warning There are annotations with the same name (but from different sets/packages). If using the wrong annotation very strange things can happen. For now we concentrate on CDI and Bean validation, i.e. all annotations should come from packages **javax.enterprise.context, javax.inject, javax.validation.constraints or javax.annotation.** (NO ...faces... package, except for faces.events, anywhere). Watch out!!

1. Now we add the model to the front-end.
2. Wrap the shop in a @Singleton This bean will keep the reference to the shop (make reference transient, i.e. lost if bean passivated, it's just for now... not optimal it's for pedagogical reasons). Add some method to access the wrapped shop. Let all beans that need a shop reference use method injection i.e. let CDI handle the initialization of references to the singleton bean.

2.3 Validation

For now there's no validation at all.

Note We try to avoid JSF validation, there should be no <f:validate..> tags in the pages.

1. Add beans Validation annotations to the backing beans. Use <h:message for="..." /> to display validation errors in pages. There are some custom validation messages in file /src/main/resources/ValidationMessages.properties. Use it to figure out some bean attribute constraints.

2.4 Authorization

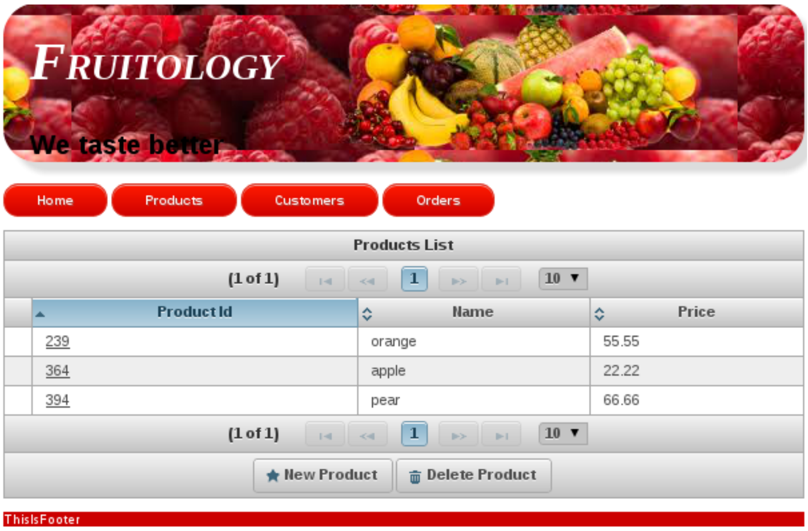
(Optional) We will use the "official" JEE style for authorization handling over a lot to the container. We also will use form based authorization.

1. First we need a user in a realm (user database). Start GlassFish Admin Console (Server > GlassFish, right click > View Domain Admin Console.
2. When started: server-config > Security > Realms > File > Manage users > New ... Add a user named qqq with password 111. Add group "products".

   Note Using a file realm is just for now, standard is to have a backing database with all users (principle the same, more later).
3. We will protect the products-pages. Inspect web.xml and add a <security-constraint>. Set <role-name>productManager</role-name>.
4. Inspect glassfish-web-xml, see <security-role-mapping>.
5. Implement missing parts of the AuthBean incl. validation and navigation. Where should we go if success, if fail?
6. This should do it... does it work (also try bypass using address field in browser)?

3 Using a JSF component suite

(Optional) For now we use simple <h: ...> components and no AJAX. If we would like to spice up the pages or create a single page applications it's better to use a higher level component suite like RichFaces, ICEFaces or PrimeFaces. Here we test PrimeFaces (visit home page to get some inspiration). Would like something like this;



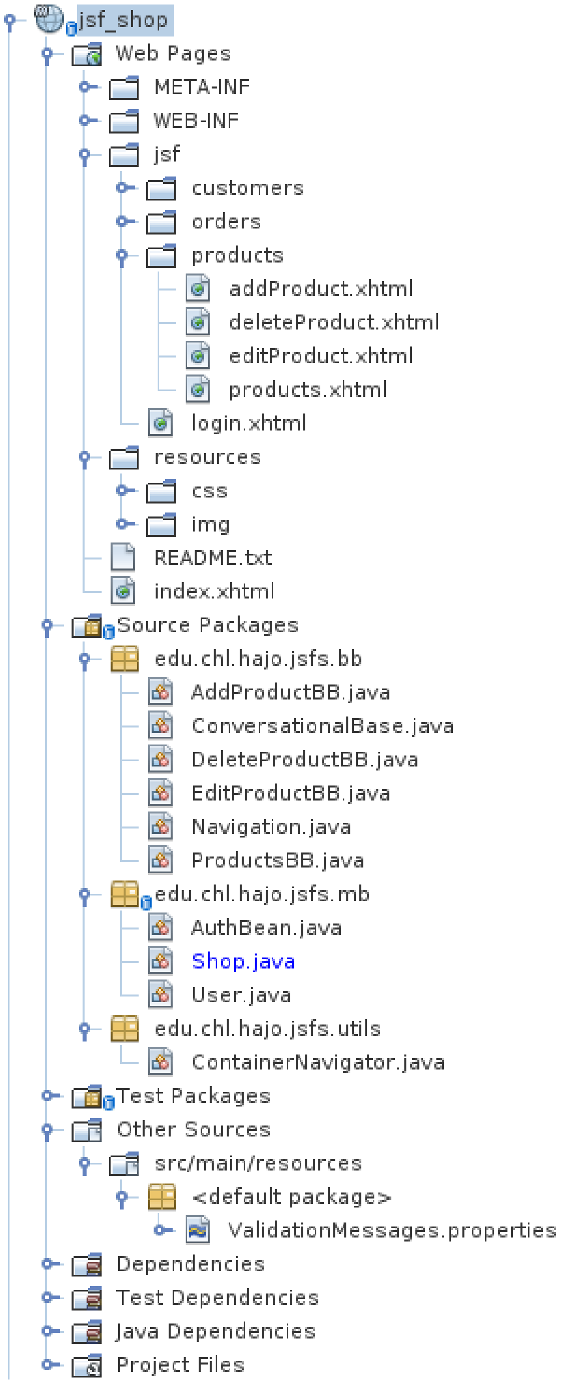
Product Id	Name	Price
239	orange	55.55
364	apple	22.22
394	pear	66.66

1. Make a clone of the previous workshop. Rename to jsf_shop_prime
2. Add support for PrimeFaces. Make sure GlassFish is selected as container. Mark project > Properties >Frameworks > Components > Check PrimeFaces > OK
3. Inspect dependencies. There' a generated file welcomePrimerFaces.xhtml. Run application and visit.
4. The goal is to replace all products pages and corresponding backing beans with a single product page and a single backing bean (similar to the REST solution). Do so. Use order.xhtml as an starting point. Copy and rename to products.xhtml Comment out <security-constraint> to disable authorization during development.

   Tip The main components to use are <p:dataTable> and <p:dialog>. Put them inside the <ui:define>.

   There's a very nice example (full version also using database, we do later...) on the course page, thanks to Emre Simtay. A lot to copy!

Appendix



```
jsf_shop
  Web Pages
    META-INF
    WEB-INF
    jsf
      customers
      orders
      products
        addProduct.xhtml
        deleteProduct.xhtml
        editProduct.xhtml
        products.xhtml
      login.xhtml
    resources
      css
      img
    README.txt
    index.xhtml
  Source Packages
    edu.chl.hajo.jsfs.bb
      AddProductBB.java
      ConversationalBase.java
      DeleteProductBB.java
      EditProductBB.java
      Navigation.java
      ProductsBB.java
    edu.chl.hajo.jsfs.mb
      AuthBean.java
      Shop.java
      User.java
    edu.chl.hajo.jsfs.utils
      ContainerNavigator.java
  Test Packages
  Other Sources
    src/main/resources
      <default package>
        ValidationMessages.properties
  Dependencies
  Test Dependencies
  Java Dependencies
  Project Files
```