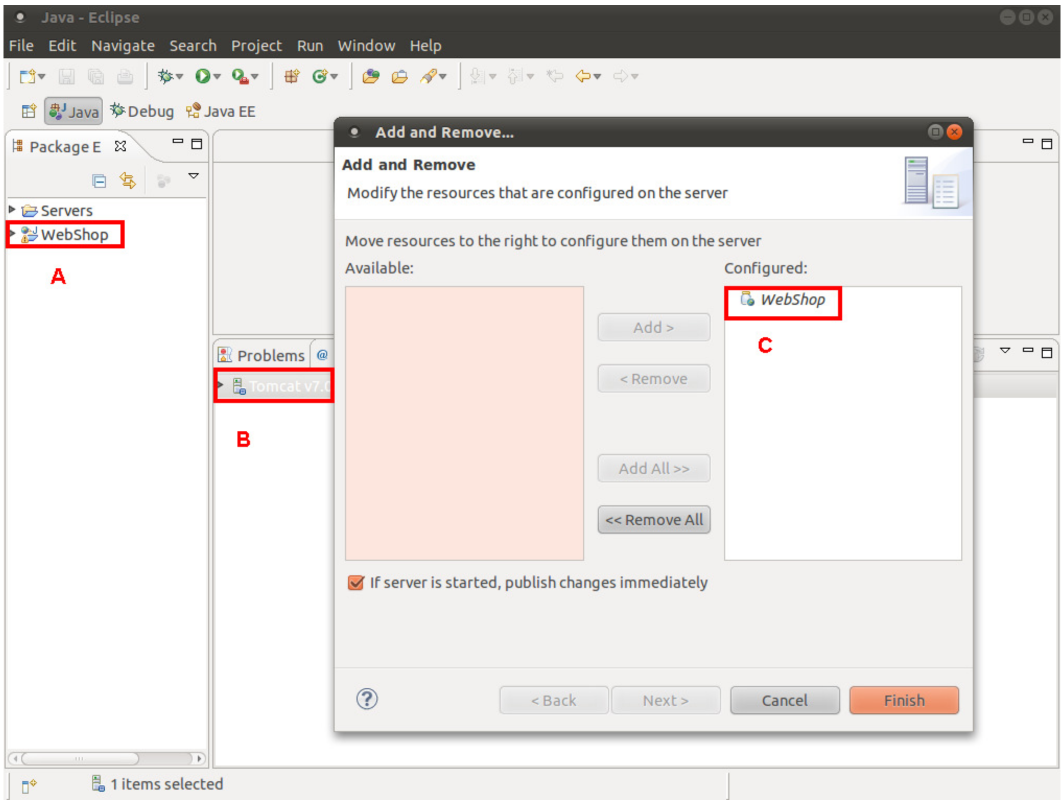
You will create an online shop with the following techstack:

- Spring 3
- Hibernate 3
- JSF2
- JSP 2.1/Servlets 3.0
- Tomcat 7
- Eclipse Helios

The application is split into 3 layers. The frontend layer is in the be.groept.web.* packages. The service layer is in the be.groept.facade.* packages and the data access layer in the be.groept.repositories.* packages. These packages are assembled in the same application for this exercise which are all deployed as a single WAR on a single tomcat7 instance. So the separation in this case not physical.

The startup project is supplied together with this document. You can extract it, and perform 'mvn eclipse:eclipse' in the root. After that you should be able to import it in your eclipse helios. You should also download tomcat 7 and configure this server in eclipse. If you don't know how to add a server in eclipse (WTP) see Toledo.

After that, you should see the imported project (A). Make sure there is a small "globe" image in the top left corner of the icon (the icon left of the project name 'WebShop'). If you look closely you'll see 3 things: a folder image, a 'J' on top of the folder image, and a globe in the left upper corner (and in my case also a warning, exclamation mark). The globe identifies your project as a dynamic web project. When you added the tomcat 7 server, you should see B. After that you can right click on the server and choose "add and remove…" you will be able to select the WebShop application (if the globe would not be there, then you would have nothing to select in the "add and remove…" dialog).


Java - Eclipse
File Edit Navigate Search Project Run Window Help
Java
Debug
Java EE
Package E
Servers
WebShop
A
Problems
Tomcat v7.0
B
Add and Remove...
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
WebShop
C
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Cancel
Finish
1 items selected

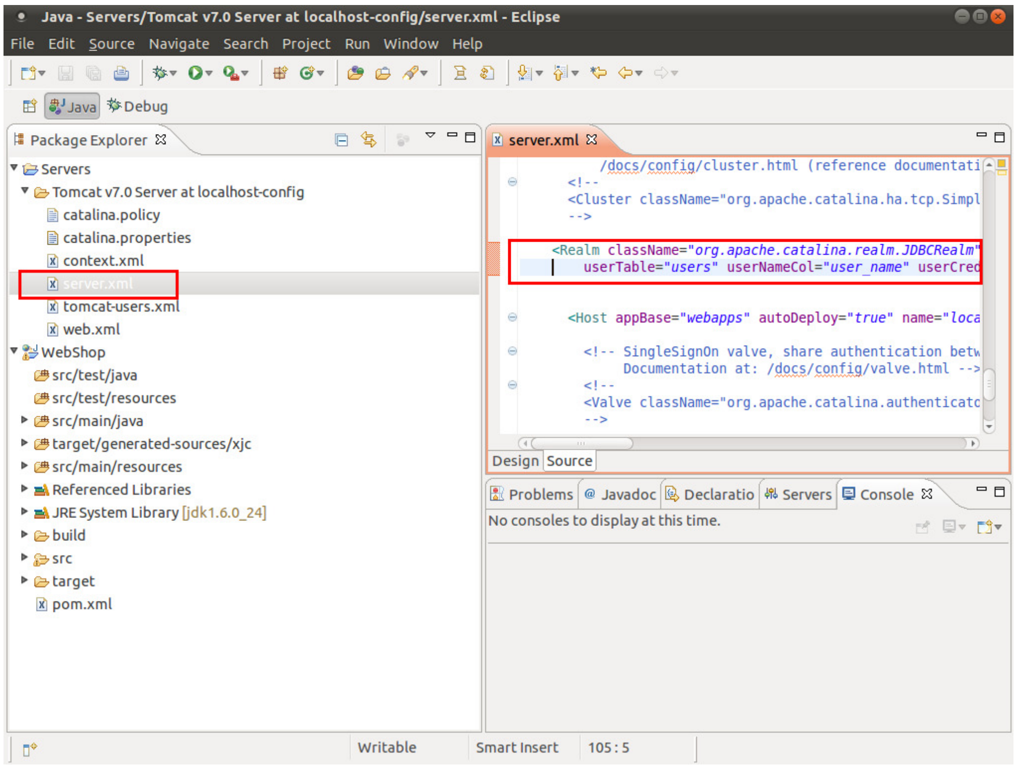
The following configuration will be required for tomcat7:

- Set the memory for the permanent generation higher
  - Double click on the “server” in the server view
  - Click the “Open launch configuration” link in the server windows
  - Click the “arguments” tab
  - In the “VM Arguments” text area add this: -XX:MaxPermSize=128m
    - Remember, add it at the end of the configuration which is already there, separated with a space. Do not remove any of the existing text in the text area
- Enable the JDBC realm. Open the server config and remove the existing realm:

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
 <Realm
className="org.apache.catalina.realm.UserDatabaseRealm"
resourceName="UserDatabase"/>
</Realm>
```
Remove this complete XML block

  - Next you add the new realm

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
driverName="org.hsqldb.jdbcDriver"
connectionURL="jdbc:hsqldb:hsql://localhost/testdb"
          userTable="users" userNameCol="user_name"
userCredCol="user_pass" userRoleTable="user_roles"
roleNameCol="role_name"/>
```


Java - Servers/Tomcat v7.0 Server at localhost-config/server.xml - Eclipse
File Edit Source Navigate Search Project Run Window Help
Java
Debug
Package Explorer
Servers
Tomcat v7.0 Server at localhost-config
catalina.policy
catalina.properties
context.xml
server.xml
tomcat-users.xml
web.xml
WebShop
src/test/java
src/test/resources
src/main/java
target/generated-sources/xjc
src/main/resources
Referenced Libraries
JRE System Library [jdk1.6.0_24]
build
src
target
pom.xml
server.xml
Design Source
Problems Javadoc Declaratio Servers Console
Writable
Smart Insert
105 : 5

```
        /docs/config/cluster.html (reference documentati
    <!--
    <Cluster className="org.apache.catalina.ha.tcp.Simpl
    -->

  <Realm className="org.apache.catalina.realm.JDBCRealm"
      userTable="users" userNameCol="user_name" userCred

    <Host appBase="webapps" autoDeploy="true" name="loca

      <!-- SingleSignOn valve, share authentication betw
           Documentation at: /docs/config/valve.html -->
      <!--
      <Valve className="org.apache.catalina.authenticato
      -->
```

No consoles to display at this time.

- You should also copy the **hsqldb-1.8.0.x.jar** to your tomcat lib directory. So, suppose your maven repository is in "c:\groept\repo\" and your tomcat in "c:\groept\apps\tomcat7", you will copy: "c:\groept\repo\hsqldb\hsqldb\1.8.0.x\hsqldb-1.8.0.x.jar" to "c:\groept\apps\tomcat7\lib"if your tomcat directory
    - Where "x" is the version number, just "10" if you have it, otherwise the highest version available. So in my case its "hsqldb-1.8.0.10.jar". If you don't have that, take the highest version which you do have.

Finally you can start the server. Output in your console should look like this:

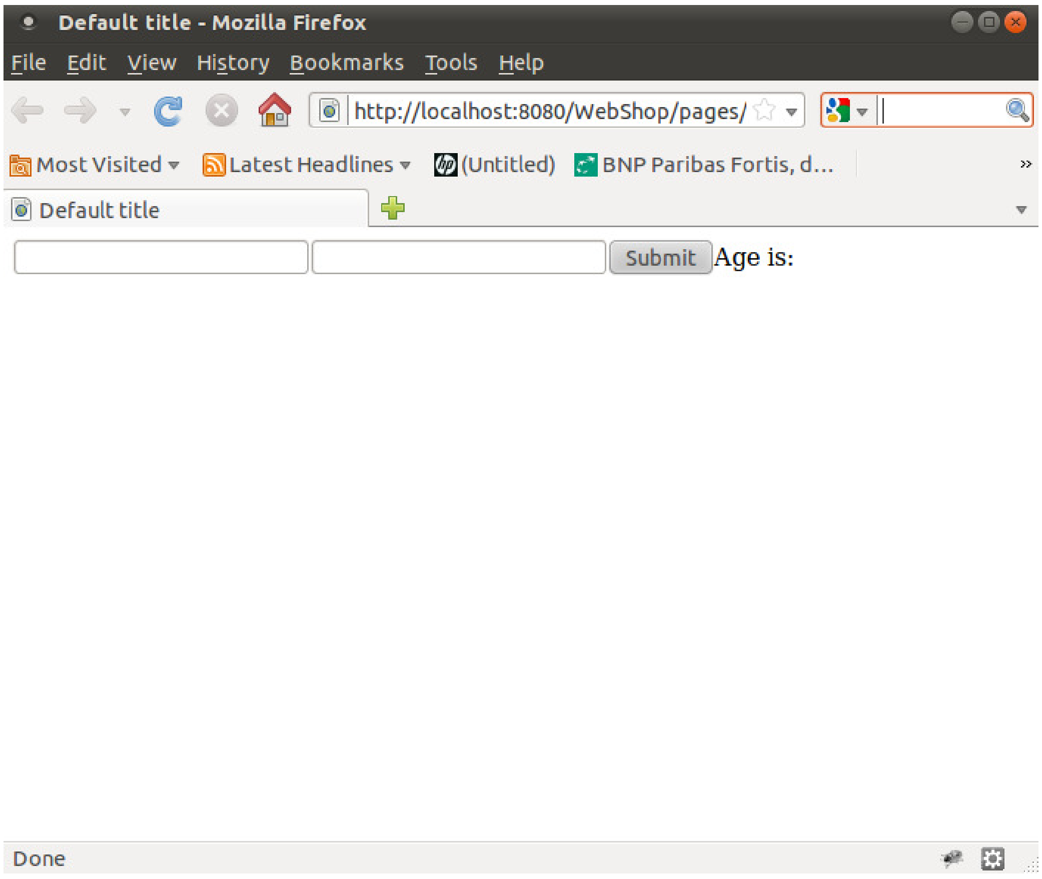
```
Aug 5, 2011 8:26:02 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The APR based Apache Tomcat Native library which allows optimal
performance in production environments was not found on the
java.library.path:
/home/koen/devel/apps/jdk1.6.0_24/jre/lib/amd64/server:/home/koen/devel/app
s/jdk1.6.0_24/jre/lib/amd64:/home/koen/devel/apps/jdk1.6.0_24/jre/../lib/am
d64:/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
Aug 5, 2011 8:26:02 PM org.apache.tomcat.util.digester.SetPropertiesRule
begin
WARNING: [SetPropertiesRule]{Server/Service/Engine/Realm} Setting property
'lazy-init' to 'true' did not find a matching property.
Aug 5, 2011 8:26:02 PM org.apache.tomcat.util.digester.SetPropertiesRule
begin
WARNING: [SetPropertiesRule]{Server/Service/Engine/Host/Context} Setting
property 'source' to 'org.eclipse.jst.j2ee.server:WebShop' did not find a
matching property.
Aug 5, 2011 8:26:02 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-bio-8080"]
Aug 5, 2011 8:26:02 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["ajp-bio-8009"]
Aug 5, 2011 8:26:02 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 788 ms
Aug 5, 2011 8:26:02 PM org.apache.catalina.core.StandardService
startInternal
INFO: Starting service Catalina
Aug 5, 2011 8:26:02 PM org.apache.catalina.core.StandardEngine
startInternal
INFO: Starting Servlet Engine: Apache Tomcat/7.0.16
Aug 5, 2011 8:26:02 PM org.apache.catalina.realm.JDBCRealm startInternal
SEVERE: Exception opening database connection
java.sql.SQLException: socket creation error
	at org.hsqldb.jdbc.Util.sqlException(Unknown Source)
	at org.hsqldb.jdbc.jdbcConnection.<init>(Unknown Source)
	at org.hsqldb.jdbcDriver.getConnection(Unknown Source)
	at org.hsqldb.jdbcDriver.connect(Unknown Source)
	at org.apache.catalina.realm.JDBCRealm.open(JDBCRealm.java:711)
	at
org.apache.catalina.realm.JDBCRealm.startInternal(JDBCRealm.java:782)
	at
org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:145)
	at
org.apache.catalina.core.ContainerBase.startInternal(ContainerBase.java:102
6)
	at
org.apache.catalina.core.StandardEngine.startInternal(StandardEngine.java:2
91)
	at
org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:145)
	at
org.apache.catalina.core.StandardService.startInternal(StandardService.java
:443)
	at
```

```
org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:145)
	at
org.apache.catalina.core.StandardServer.startInternal(StandardServer.java:7
27)
	at
org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:145)
	at org.apache.catalina.startup.Catalina.start(Catalina.java:620)
	at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:3
9)
	at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImp
l.java:25)
	at java.lang.reflect.Method.invoke(Method.java:597)
	at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:303)
	at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:431)
Aug 5, 2011 8:26:21 PM org.apache.catalina.core.StandardContext
addApplicationListener
INFO: The listener "com.sun.faces.config.ConfigureListener" is already
configured for this context. The duplicate definition has been ignored.
Aug 5, 2011 8:26:21 PM com.sun.faces.config.ConfigureListener
contextInitialized
INFO: Initializing Mojarra 2.1.3 (FCS b02) for context '/WebShop'
Aug 5, 2011 8:26:21 PM com.sun.faces.spi.InjectionProviderFactory
createInstance
INFO: JSF1048: PostConstruct/PreDestroy annotations present.  ManagedBeans
methods marked with these annotations will have said annotations processed.
Aug 5, 2011 8:26:22 PM org.apache.catalina.core.ApplicationContext log
INFO: Initializing Spring root WebApplicationContext
[Server@2c5a7942]: [Thread[Thread-2,5,main]]: setRestartOnShutdown(false)
[Server@2c5a7942]: [Thread[Thread-2,5,main]]: setNoSystemExit(true)
[Server@2c5a7942]: [Thread[Thread-2,5,main]]: checkRunning(false) entered
[Server@2c5a7942]: [Thread[Thread-2,5,main]]: checkRunning(false) exited
[Server@2c5a7942]: Initiating startup sequence...
[Server@2c5a7942]: Server socket opened successfully in 1 ms.
[Server@2c5a7942]: Database [index=0, id=0, db=mem:testdb, alias=testdb]
opened sucessfully in 83 ms.
[Server@2c5a7942]: Startup sequence completed in 86 ms.
[Server@2c5a7942]: 2011-08-05 08:26:22.874 HSQLDB server 1.8.0 is online
[Server@2c5a7942]: To close normally, connect and execute SHUTDOWN SQL
[Server@2c5a7942]: From command line, use [Ctrl]+[C] to abort abruptly
Aug 5, 2011 8:26:23 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Aug 5, 2011 8:26:23 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Aug 5, 2011 8:26:23 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 20884 ms
```

The exception in red is normal. We start our internal database when the server is started. At the time the server is starting, our JDBC Realm is trying to connect to the database, which is not yet there. This does not matter, since it will re-connect later when we actually use the JDBCRealm and then the database will be booted and no further errors arise.

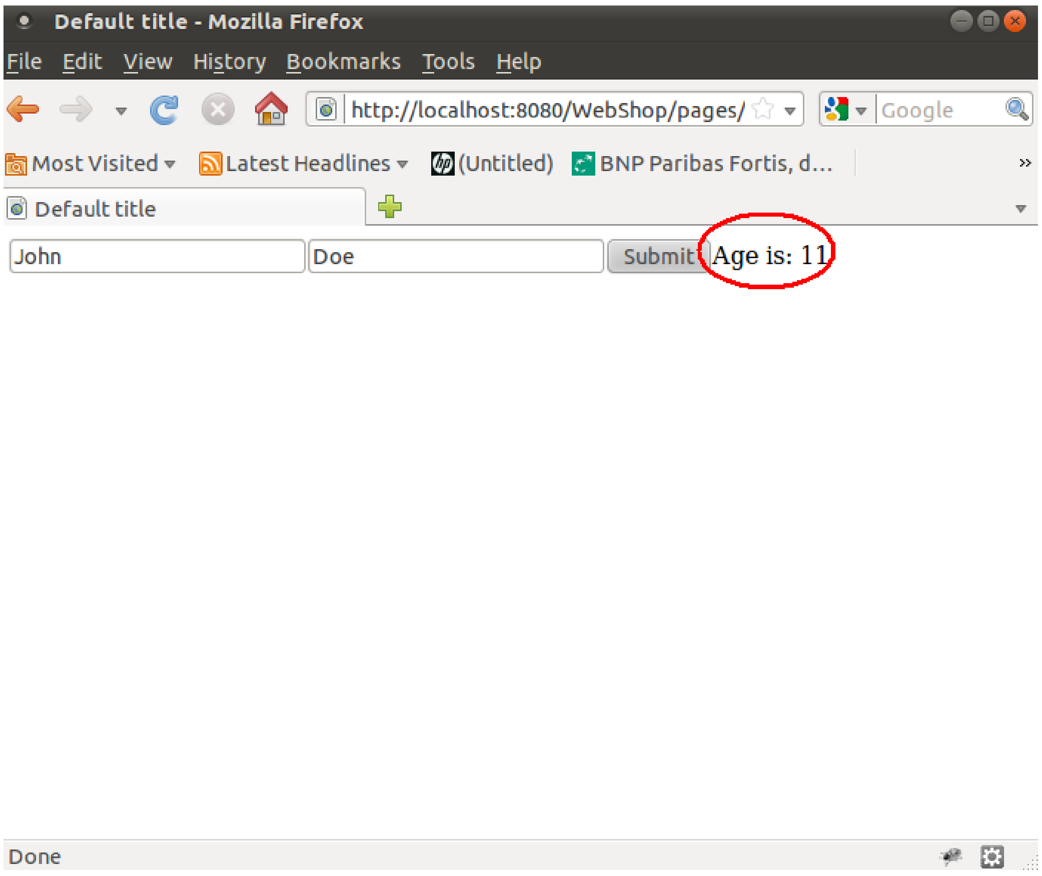
Next you can point your browser to:

http://localhost:8080/WebShop/pages/secured/menu.faces


Default title - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebShop/pages/
Most Visited
Latest Headlines
(Untitled)
BNP Paribas Fortis, d...
Default title
Submit
Done

Age is:

When you enter ‘John’ and ‘Doe’ and hit submit:


Default title - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebShop/pages/
Google
Most Visited
Latest Headlines
(Untitled)
BNP Paribas Fortis, d...
Default title
Submit
Done

John

Doe

Age is: 11

When you see all of this, the startup application is fully functional. You can now use it to start your development. All the base classes which are there (the backing action, the façade, the repository and the entity) you can let them as they are (you may remove them, or leave them be for reference if you need them).

Here are the rules you need to follow:

- You should continue to follow the same principles as already available in the startup app. This means that you should continue using the 3 tiered separation, by using backing actions to capture your frontend events, facades for you business logic, and repositories to manager you data access.
  - Facades and repositories are defined as Spring beans. There is no autowiring allowed there, so you should use the XML configuration to pass along dependencies
  - You should use XML configuration for you hibernate entities, no annotations. The startup entity serves as an illustration.
  - Backingactions talk to facades and do not perform direct data access (it are the facades which on their turn talk to loaded entities and/or repositories for data access). As the startup app demonstrates you are allow to inject your facades using autowiring into the backing actions
    - New components that you must write, such as JSF converters or JSF validators, or XML parsers etc, you are free to create extra packages are you like. Just make sure they belong to the correct tiered base packages. Example; JSF validators are for the frontend, so its clear they will belong somewhere in the "be.groept.web.*" package. Maybe a sub package like: "be.groept.web.validators".
- When the application starts, an in-memory database is started along sides. All data setup must be put into the "TestDataSetup" class. The setup will make sure that the data you setup here is saved into the database. Note, that when you restart the app, any modifications you did to the database manually are lost. After a restart the database is reset with the data which is setup in the TestDataSetup.
  - You ARE allowed to connect to a STANDALONE HsqlDb instead of the in memory db (we did this in class with the glassfish setup). You can change the database URL easily in webshop-persistence.xml. This might be easier to develop, because if you restart the application the database restarts with it. If you have a JDBC client open to the database, it will lose its connection and not reconnect automatically (this happens with for example squirrel). If you have a standalone database you will not have this as it keeps on running even if you restart your webapp. **However:** when you submit your solution at the end, the database configuration should be reverted to the in memory variant, so that when I test your application everything runs out of the box. Also, your app should be populated with data and run without any modifications! So if you entered data manually in your stand alone database but did not add them to the TestDataSetup, then that data will not be there if you switch the configuration back. So be careful!
- You should only use libraries which are put on the classpath by Maven. Everything you need is already there, there is no need for additional libraries.
- The text in the webapplication needs NOT to be localized. So you can put static text directly in the XHTML pages

The application should supports 2 roles:

- Registerered
- Super

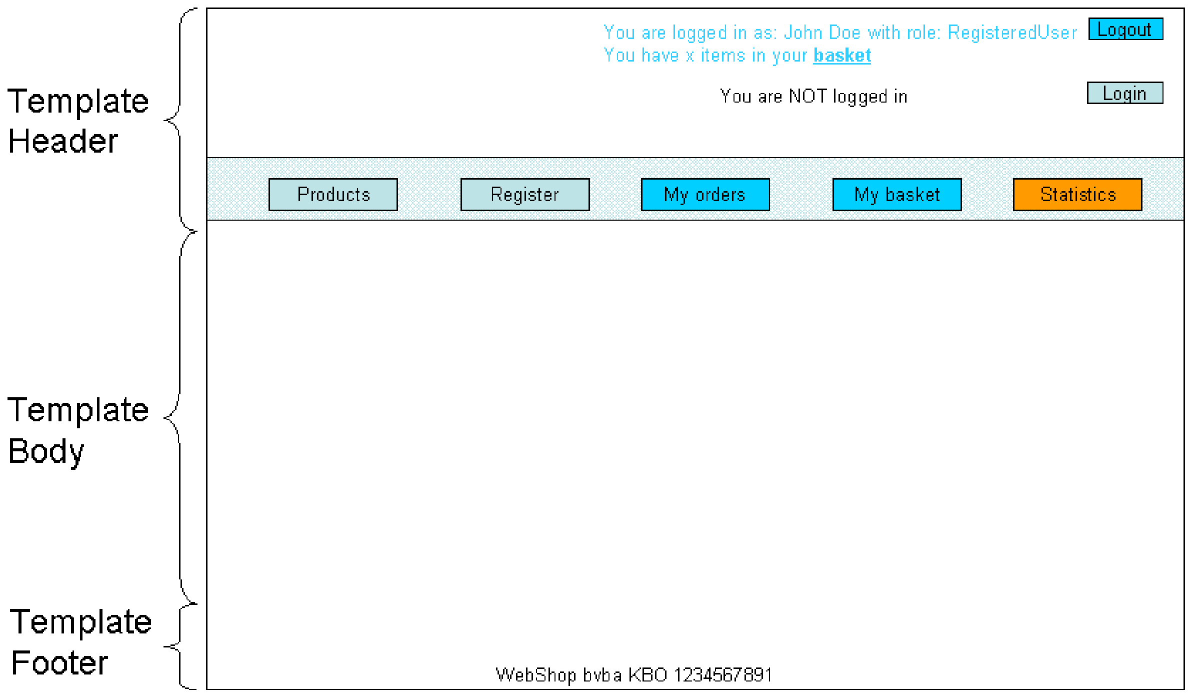
When a user enters the site, he is able to browse around without being authenticated. When he wants to place an order, the user should login first using the login button. After logging in, the user will see extra buttons and can for example can add orders to the basket and perform a checkout. Super users can do the same as registered users, but then can access some more functionality such as generating statistics.

You will use a form based login page which the user can use to enter the username and password credentials. The menu page will allow a anonymous user to create a new account as a registered user. To make it easy, you can show a checkbox so the user can choose if he wants to be a normal registered user or a super user (not very realistic, but this will do fine).

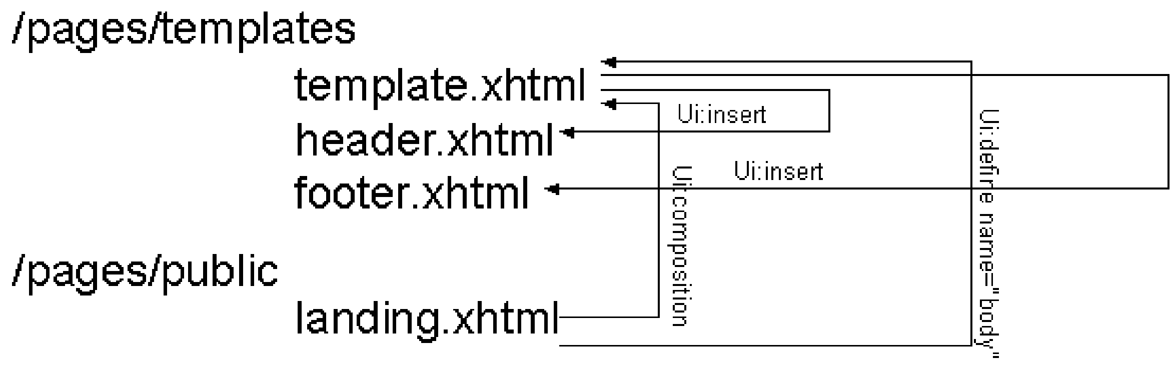
When the user opens the application the landing page is shown Remember, NOT the login page is shown, since this application allows unauthenticated users to browse the site (you can also browse around on amazon without having to login first). The landing page is accessible for everyone, so you should put it under pages/public/. The rule is that

- Public template pages are put in pages/template
- Public pages are put in pages/public
- Private pages (for which authentication is required) are put in pages/private

The landing page will look like this:



You have to create a template that each page implements. The template consists out of a header, a body and a footer. The header and footer are thus re-used. The body you will implement for each page.

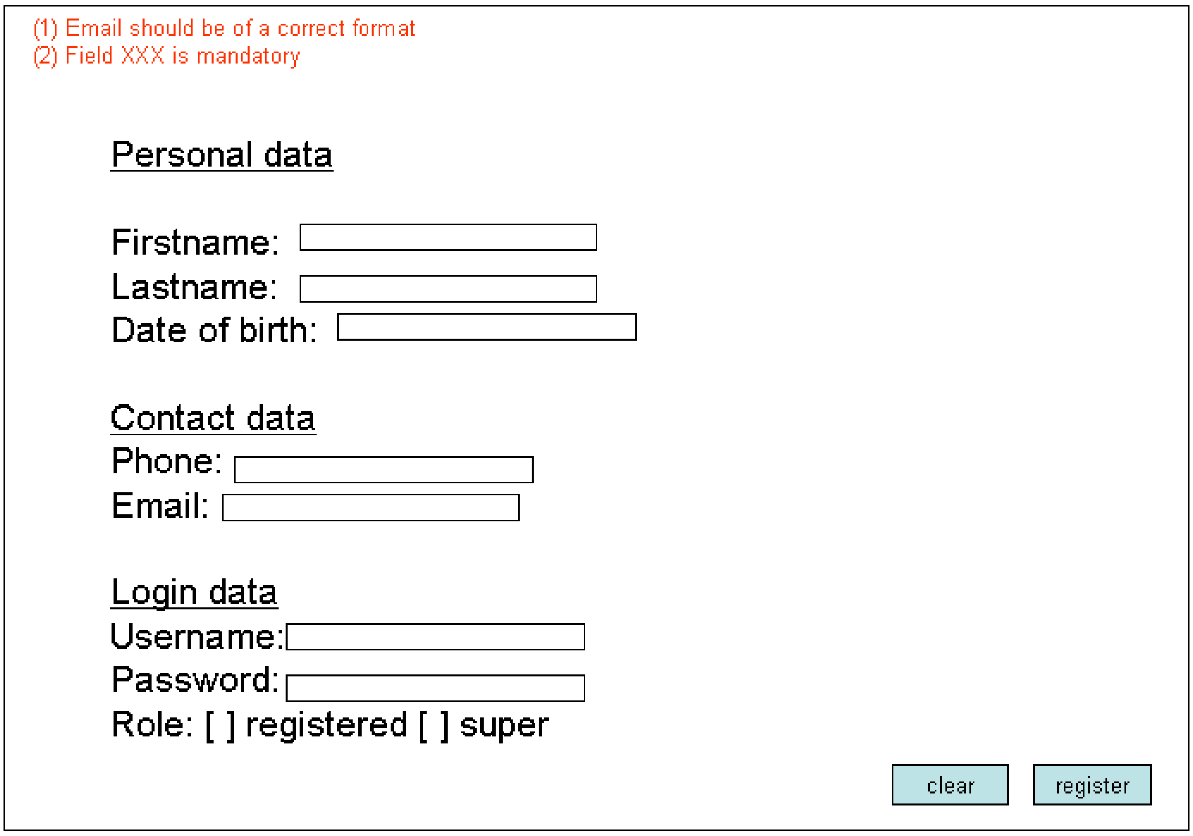


The landing page implements the template but has an empty body, as you can see. The other pages, such as "Products" will define a body as shown on the next pages. **Remember, on the next pages I'll only show what should be placed inside the BODY content. But EACH page has the header, a body and a footer as illustrated above on the menu page via the templating**. This allows you to logout from every page, or use the navigation bar from every page.

The colors in the menu page show the buttons and/or text that is displayed depending on the role the user has. If the user is without a role, not authenticated, then only the green buttons and black text + green login button is shown. If the user logs in, and enters a username/password from a user with "registered" role access, then the buttons and text in blue become visible. Note that the buttons in green remain visible (a registered user can also browse products), but the text in black "you are not logged in…" disappears and the text in blue appears. If the user enters a username/password of that of a user in "super" role, then the blue buttons appear, the black text and logout button disappears (as with the user in "registered " role) but also the yellow button appear. Als note that the "**basket**" is a link. It will navigate to the "my basket" page (it is a shortcut, so clicking 'basket' link or the 'my basket button' results in the same page being shown).

The pages: my orders, my basket and statistics are all secured page and should be placed in /pages/secured. The landing page, products page, register page and login page are placed in public. Note that the only security we apply here is hiding of the buttons if a user is not a given role. So for example; if I login with a user in "registered" role, then the button "statistics" is not shown. However, nothing withholds me from going to the URL directly, or simulate the http request that happens when you would have had the super role granted. Normally extra security mechanisms are in place to make sure you cannot access any functionality in the secured directories, not even with direct http access. However, for the scope of this application we don't configure this any further (we just do the "visual security" by not showing the buttons if the user has not sufficient authorization).

The register screen


(1) Email should be of a correct format
(2) Field XXX is mandatory

Personal data

Firstname:
Lastname:
Date of birth:

Contact data
Phone:
Email:

Login data
Username:
Password:
Role: [ ] registered [ ] super

The register screen will register a user, and save the information in the database. This is a special situation. For all other screens you are free to setup the datamodel as you whish, however for the registration the tables and columns structure and names must match the values we used in the JDBC realm in the tomcat configuration, which we did on the first page. Tomcat will automatically use these tables to check the authentication if a user logs in.

```
userTable="users"
userNameCol="user_name"
userCredCol="user_pass"
userRoleTable="user_roles"
roleNameCol="role_name"
```

This means you will have to create an entity with table “users” and two columns “user_name” and “user_pass”. You will also need an entity with table “user_roles” and two columns “user_name” and “role_name”. Note that each table is allowed to have surrogate key as its primary key (remember that we use surrogate key for each table). You can make the relation based upon the surrogate (primary) key. So for example, you could create an entity “User” and an entity “UserRole”. A User has one or more UserRoles. So in the User entity you will have a “Collection<UserRole>”. In the hibernate mapping file you will probably map this as a “<bag>” on the User side. This enables you to store users and roles in the database, and by using the same table names and columns as we configured the JDBC realm with, tomcat is able to read our stored user data from the database. So, if I register a user here, after hitting “register” I should be able to login directly afterwards.

Note: hibernate will create tables and columns automatically based upon your mapping files. This is configured by the “<prop key="hibernate.hbm2ddl.auto">create</prop>” in the webshop-persistence.xml. So, to make your testing easier, after you developed the registration page (and after it works) you can add some user data to the TestDataSetup class, so that after the tables are created user data is inserted which you can use to test.

- The error messages shown are generated when any of the fields are empty (all fields are mandatory) or the email address is of a wrong format. Use a validator to check the email format, it should be (minimal) something like: a@b.cd

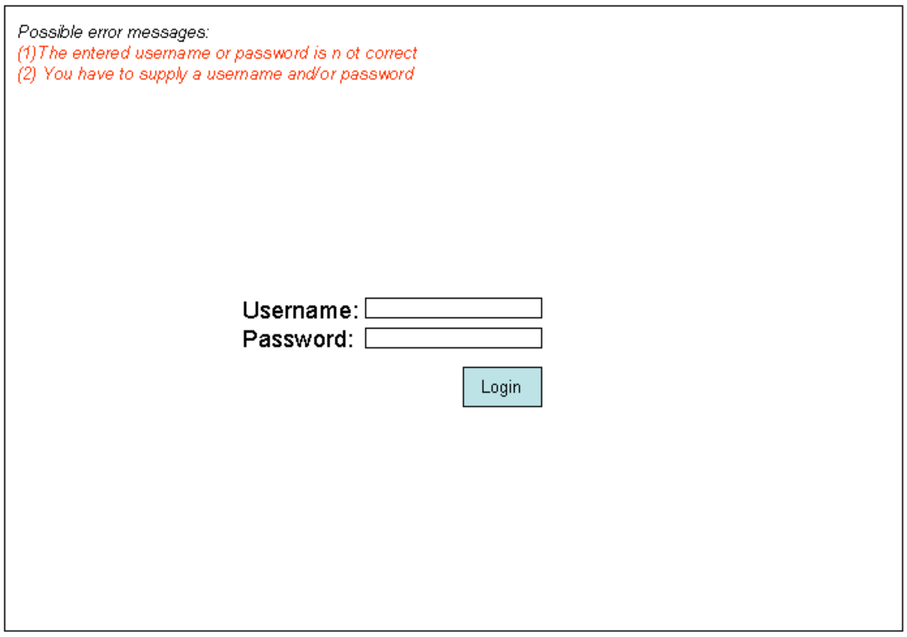
Also, you'll need to add extra information to the web.xml. For example:

```
<login-config>
      <auth-method>FORM</auth-method>
      <realm-name>Webapp</realm-name>
      <form-login-config>
            <form-login-page>/pages/public/login.faces</form-login-page>
            <form-error-page>/pages/public/login.faces</form-error-page>
      </form-login-config>
  </login-config>

  <security-role>
            <role-name>registered</role-name>
            <role-name>super</role-name>
  </security-role>

<security-constraint>
      <web-resource-collection>
            <web-resource-name>Secured zone</web-resource-name>
            <url-pattern>/secured/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
            <http-method>HEAD</http-method>
            <http-method>OPTIONS</http-method>
            <http-method>PUT</http-method>
            <http-method>TRACE</http-method>
            <http-method>DELETE</http-method>
      </web-resource-collection>

      <auth-constraint>
            <role-name>registered</role-name>
            <role-name>super</role-name>
      </auth-constraint>
  </security-constraint>
```

Note: when you now access a secured resource directly via the URL (in the /pages/secured/* folder) you will get a error page if you are not yet logged in. You will not be automatically forwarded to the login page, this is normal and good as is. When you however login first, and then access a secured page by clicking on the newly appeared buttons enabled for the role of the logged in user, in your menu (example "my orders") the page should be shown without problems.

<u>The login screen</u>

If an unauthenticated user hits the login button, this page is shown:


Possible error messages:
(1)The entered username or password is n ot correct
(2) You have to supply a username and/or password

After logging in, the user is redirect back to the **last page**. For example, if the user was browsing in the products, and hits “login” on the header, the user should be redirected back to the products page. The buttons/text become visible depending on the role the user was authenticated with. The text in red are validation messages and they are shown when the user did something wrong. Entering wrong username/password or supplied no username or password. In that case the navigation does not occur and the login page remains shown, but the error messages are displayed in the top left corner, as shown on the previous image.

The login backing action which captures the username and password and logs in, can look like this (taken from the JSF project in the course, which we completed during classes):

```
@ManagedBean(name = "loginBackingAction")
@SessionScoped
@Controller
public class LoginBackingAction {

      private String username;
      private String password;

      public String logon() {
            HttpServletRequest request = (HttpServletRequest) FacesContext

      .getCurrentInstance().getExternalContext().getRequest();
            try {
                  request.logout();
                  request.login(username, password);
            } catch (ServletException e) {
                  return "error";
            }
            return "success";
      }

      ...
      ..
      .
```

This will take the username/password which you entered on the page and submit it to tomcat. Tomcat will use the JDBC realm to query the tables and check if username /password is correct. In your pages you can then use "`HttpServletRequest.userInRole("")`" to programmatically check if a user has a certain role (this will of course always return false if a user is not logged in). You can access this class via an implicit object (named "request") in EL. Example: "<h:button rendered="#{**request.userInRole**("registered"}" value="My orders"/>

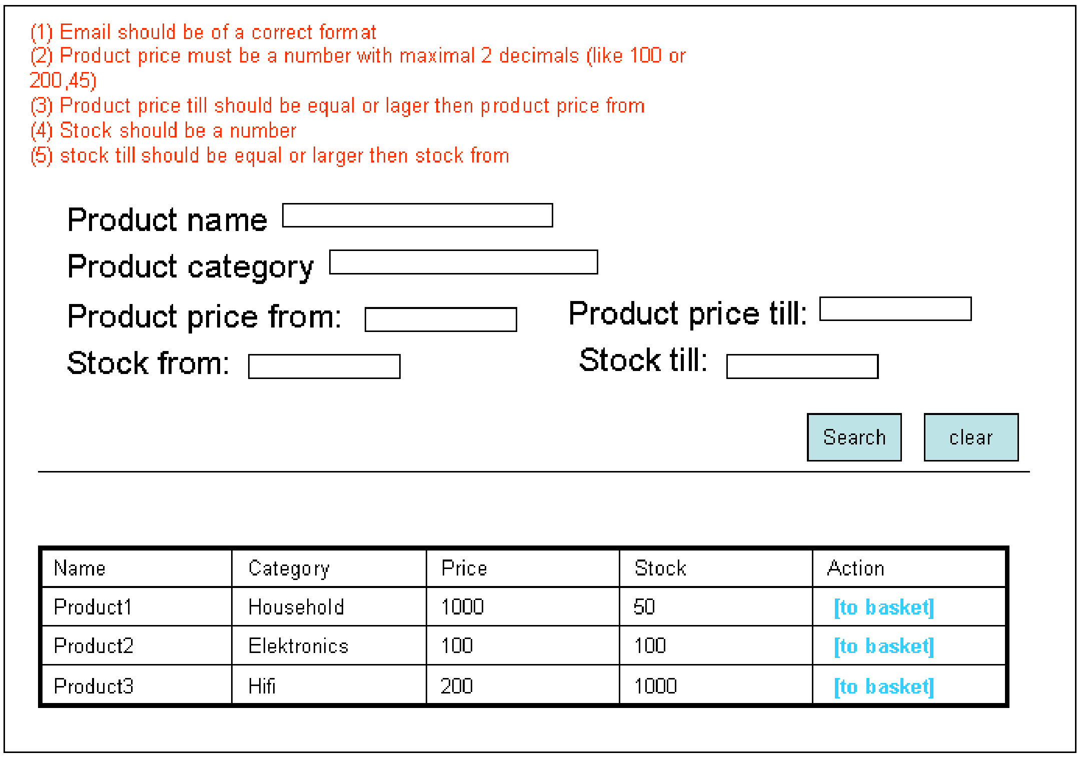
<u>**The products screen**</u>

The products screen allows users to search products. Remember that every user (without having to login) can search for products. The results (if any) are shown in a datatable (you can use h:dataTable for example, like we did in the JSF project during classes). For each product that matches the search criteria, you can order them. But only if you are logged in and are (at least) registered user (remember, "super" users can also order).

- You can choose whichever product categories you want. Just make sure you setup some test data with a couple of them so you can easily test that your app is working
- The price from/till means that only products within that price range are shown. If only price from is filled in, only products having exact that price are shown
    - The same principle for stock
- If the users hits "to basket" a single product is added to his basket. So if the users clicks 10x the same product, their will be 10 of those products in his basket. Also note that the stock
    - When a product is in the users basket, the "stock" of the product should be adjusted. So it should go down
        - Later when the users discards its basket, the stock number should be increased again with the released (non ordered) products.
    - Note that this basket is a special basket, in that sense that the information is stored in database. So if a product is added to a basket, the quantity in stock of that product is diminished by 1, as if the order was actually ordered. However, as long as a product is in the basket, no 'Order' is created. There is just a link between the users's basket and the product. So if I the user would logout and lgin, the basket would be retrieved and still shows what the user placed into the basket during his previous session

You should read/store all of this in database. The data model is up to you to design and implement. It is clear that we have at least following entities: Product, Category, Basket. There will be a relation between basket and the user table we created before (afterll, a basket belongs to a user) etc.

All product and category data is added to the application via the TestDataSetup class. There are no pages foreseen to add products and or categories manually.

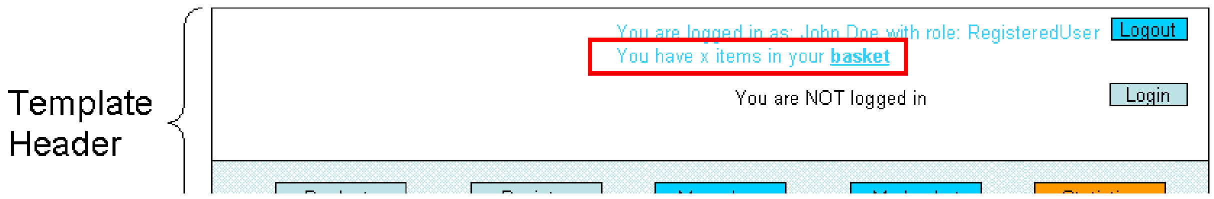


(1) Email should be of a correct format
(2) Product price must be a number with maximal 2 decimals (like 100 or 200,45)
(3) Product price till should be equal or lager then product price from
(4) Stock should be a number
(5) stock till should be equal or larger then stock from

Product name	
Product category	
Product price from:	
Product price till:	
Stock from:	
Stock till:	

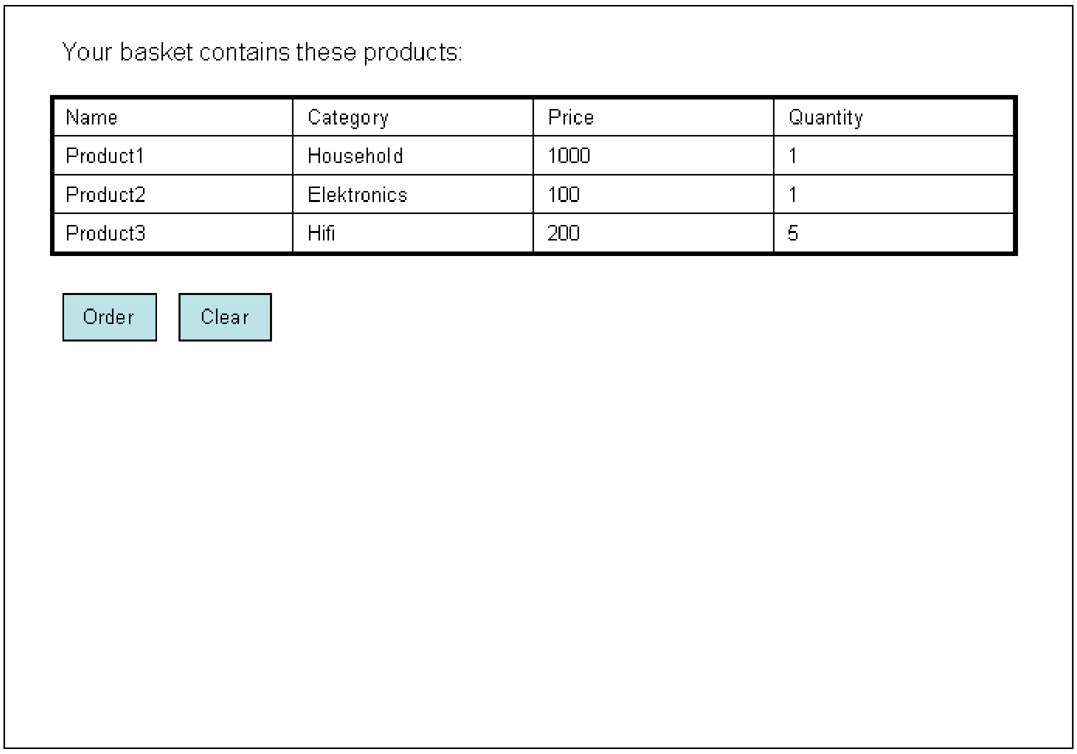
Name	Category	Price	Stock	Action
Product1	Household	1000	50	[to basket]
Product2	Elektronics	100	100	[to basket]
Product3	Hifi	200	1000	[to basket]

Also do not forget about in the header (which we don't show here, but which is always there because of the template) which contains an indication of the amount of items in the basket! This number should increase after you added a product in the basket:



You are logged in as: John Doe with role: RegisteredUser
You have x items in your basket
You are NOT logged in

<u>The my basket screen</u>



Your basket contains these products:

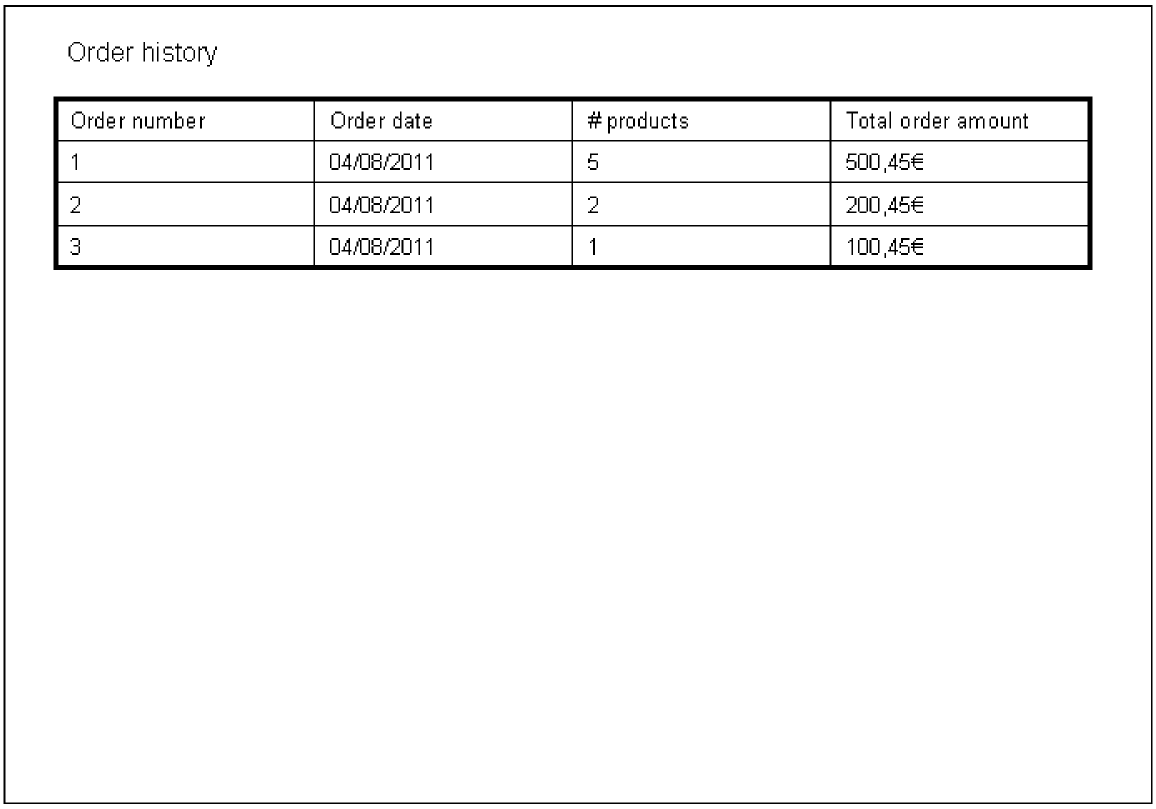
Name	Category	Price	Quantity
Product1	Household	1000	1
Product2	Elektronics	100	1
Product3	Hifi	200	5

The basket screen allows the user to create an order from items in his basket. When there are no products in basket an empty table is shown.

- When the users hits ‘order’, a new order is made. The order total is calculated based upon the products the user ordered. The order should also RETAIN a link to the products that ordered. So the “# products” that are ordered within the order is NOT stored on the order. The order has a link to the products which are part of the order and does a live count.
    - So you will probably have a many-to-many relation between product and order. A product can belong to multiple orders and an order can have multiple products
- In the same transaction where you create the order, you should also CLEAR the persistent basket of the user in the database

<u>The my orders screen</u>

Shows the history of orders made by the customer. If no orders are there, an empty table is shown.


Order history

Order number	Order date	# products	Total order amount
1	04/08/2011	5	500,45€
2	04/08/2011	2	200,45€
3	04/08/2011	1	100,45€

- **Again** the “# products” that are ordered within the order is NOT stored on the order. The order has a link to one or more products. So in your (Order) entity you will probably have a: Collection<Product>. The “size” of the collection will be the number of products belonging to that order
- The s same goes for the total order amount. You calculate that on the fly by iterating over the orders and making the sum of the price of each product multiplied by the quantity that was ordered for that product

<u>Statistics page</u>

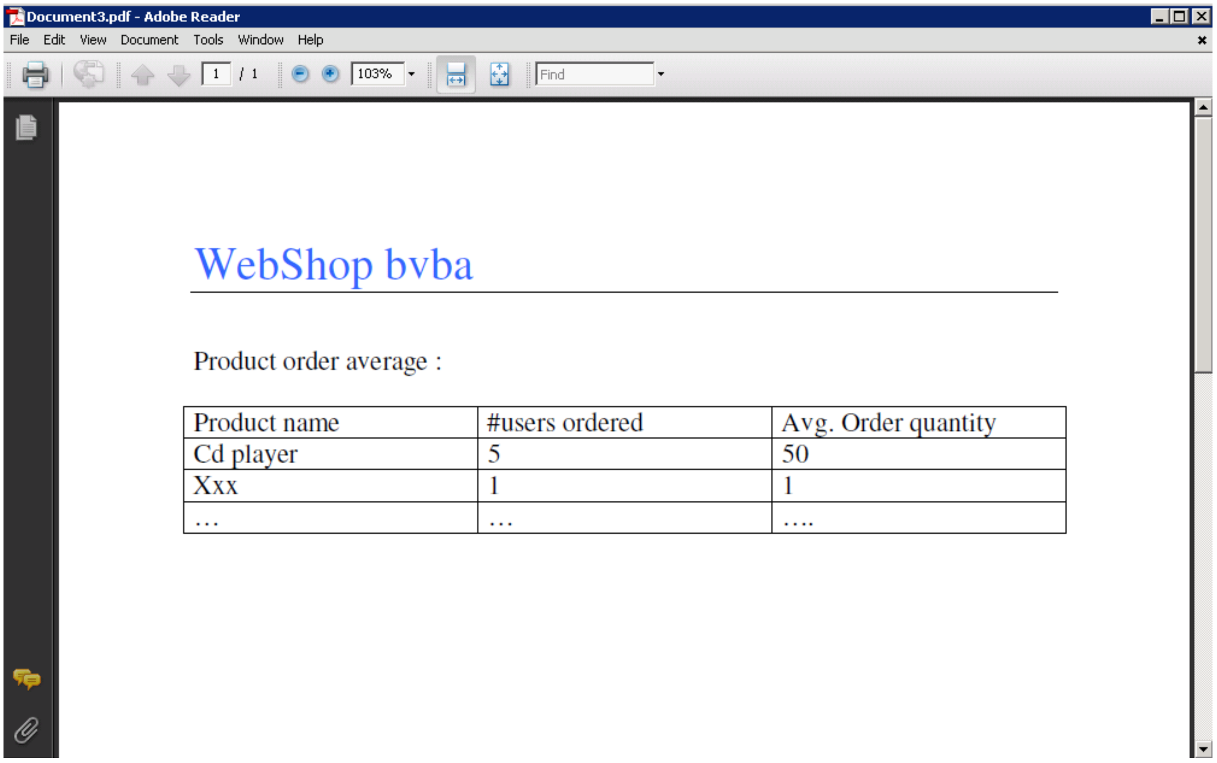
The statistics page generates a PDF file with statistics about the product/order/user relation. The PDF may be generated with jasper reports OR with FOP. If you use jasper reports, you can put the jrxml files in src/main/jasperreports. They will be automatically compiled when you perform “mvn process-resources”. In the XML processing course we spended some time on how to generate this reports. There were also some excercises done in class, so if you refer to them everything should be fine.

- Important note: if you use FOP: there is NO XML and NO XSD in this case. You can however pass along variables to you XSL document. So you are allowed to pass along the dynamic information from java to the XSL, which then shows the values. If you use jasperreports, you will probably want to the se JavaBeanCollectionDatasource (or something like that)

The statistics that are calculated are pretty easy:

- For each product that is at least ordered once (so its at least present in one order) you calculate the number of users that ordered that product (2nd column) and the average order quanity (3th column). So, if 5 users ordered in total 250 cd players, the avg order quantity is 50 (250/50)

- Remember, this data is DYNAMIC, so if I order a new product, and generate the statistics again, this new product should appear on the pdf

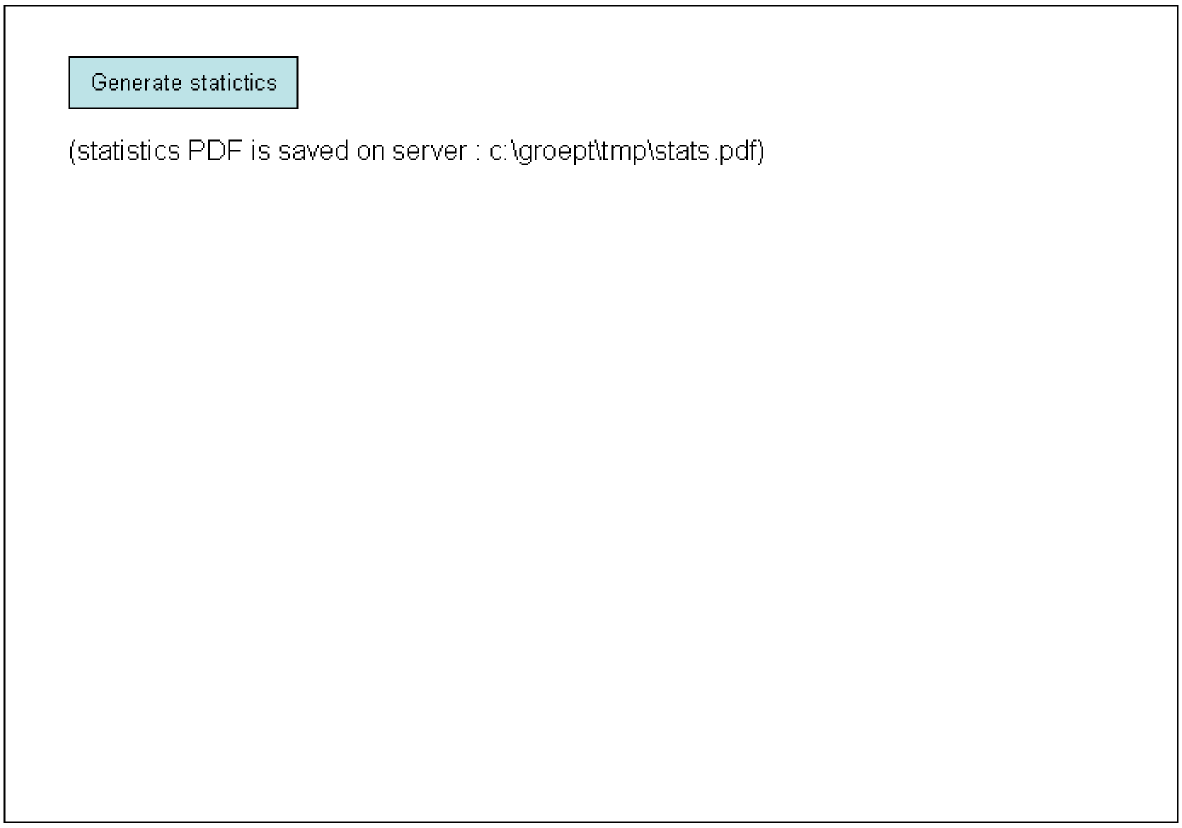

Document3.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
103%
Find

WebShop bvba

Product order average :

Product name	#users ordered	Avg. Order quantity
Cd player	5	50
Xxx	1	1
…	…	….

The PDF may be generated on the file system of the server somewhere. Indicate on the page where it is saved. You are allowed to extend this so that the PDF is actually returned via the browser, this is however <u>not required.</u>



(statistics PDF is saved on server : c:\groept\tmp\stats.pdf)