

MyPetstore Development Notes (Overview)

Overview:

The MyPetstore application is one that I found on the internet ([link](#)) and adapted for use in NetBeans. It uses JSF, Hibernate, and Spring, as well as a back-end database connection to MySQL. All in all, a nice fit for the course!

Our initial work with the application will be to get to know it and see how the pieces fit together. In the next class, we'll consider how it uses the Spring framework to make the interaction between the application components cleaner, more maintainable, and extensible.

Another key benefit of this application is that it's implemented using several enterprise application patterns (e.g. ServiceLocator, ServiceInterface, DAO). For now, we'll notice these patterns, which make the design and implementation of large projects more robust, and discuss them in more detail in the next class session. A high-level view is below:



Running the application:

To run the application, do the following:

1. Ensure that MySQL is started.
2. In the Runtime window, select Databases, then right-click the "jdbc:mysql" connection, select Connect, and enter the msis531 password ("g3tm3data"). Check the "remember password" box and click OK to connect.
3. In the Projects window, right-click the mypetstore application and select Run Application. If you see the parrot, everything's working well.



Project Structure:

The project structure follows the NetBeans standard, rather than the "non-NetBeans" ones we saw in the Hibernate section. The initial download from the internet did not follow the NetBeans structure, so I thought it would be a difficult migration, but it turned out to be straightforward once I figured out where everything needed to go. The knowledge of how to do this migration is not useful for the class, but as a project manager or implementer, it's worth knowing that migrations are possible, if, for example, your company changed Java IDEs.

The "Web Pages" section of the project has a different structure than we've seen previously: there is a set of .jsp pages at the root of the tree, and a "tiles" directory containing an identically-named set of pages. This structure allows the use of a technology called Struts Tiles (discussed below) to make the setup and maintenance of the view tier more consistent. The "images" folder contains the various images used by the application: references to these images are stored as HTML tag text in several database tables (bannerdata, category, product).

The "Configuration Files" section has more files in it than we've seen before: we'll look at some of these files now, and leave others for our session on Spring. For now, note that there are several

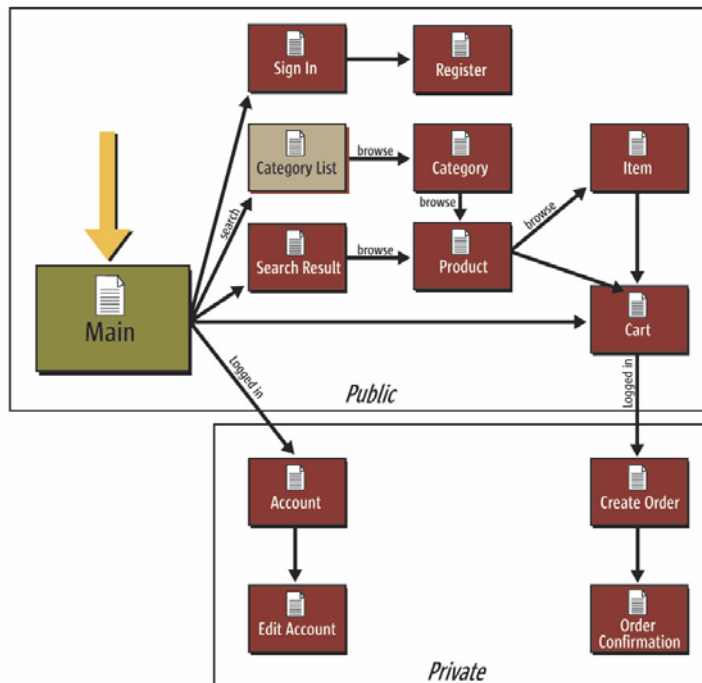
“faces-*.xml” files, which separate the various JSF functions by file. This is a nice improvement over what we’ve seen previously.

The “Source Packages” folder is subdivided using Java packages into “model” and “view” tiers, and within those, by subcomponent. A list of the packages and their purposes is shown below:

Package	Purpose
default	Store log4j (application logging) properties
model.businessobject	Hibernate mapping files and Java classes
model.businessobject.dependentobject	Objects depending on model.businessobject objects
model.dao	Data Access Object (DAO) sources (interfaces)
model.dao.sequence	Sequence-generating JavaBean
model.dao.hibernate	Implementations for model.dao
model.exception	Application-specific application handler
model.service	Service pattern interfaces
model.service.impl	Service implementations
view.bean	POJOs corresponding to view (JSF) pages
view.bundle	Message bundles
view.service	Service “finder” interface (Spring magic happens here)
view.util	JSF utility classes, security (filter) class
view.validator.regex	Regular expression processor

Application Navigation:

The navigation graph for the application is shown below:



We’ll surf around a bit in the application to try out all of this functionality, which mirrors the steps found in standard e-commerce sites.

Tiles setup:

Tiles is a technology for abstracting the standard components in a view tier, allowing easy reuse without significant effort. From a historical perspective, it's fun to see Tiles included in this application, since it was designed as part of a competing (arguably "parental") technology to JSF, called Struts. The MyPetstore author has had to hack things a bit to make it fit (since JSF does not have built-in Tiles support), but the result is very nice.

To get a sense of how Tiles is used, look at the following files in the project:

1. Under Web Pages, open WEB-INF/web.xml, and click on the XML tab. Near the bottom, the Tiles Servlet (yes, just another servlet with special powers) is defined. The configuration file for tiles is noted as tiles.xml.
2. Open tiles.xml in the same folder as web.xml. Note the name attribute (".mainLayout") and the path (to "/layout.jsp"). The "put" tags associate a name (e.g. "footer") with a JSP page (e.g. "/tiles/footer.jsp").
3. Open layout.jsp in the root of the Web Pages folder. Note this is a JSF page, but one with an additional tag library (prefix="tiles") at the top. Within the JSF page, you can see <tiles:insert> tags used to drop in the files referred to in tiles.xml. The outermost JSF tag is an <f:view> tag, which we've seen before.
4. Open category.jsp in the root of the Web Pages folder. This page is straight JSP (Tiles), no JSF taglibs at the top. This page includes the associated page in the tiles folder.
5. Open category.jsp in the tiles directory of the Web Pages folder. This page is straight JSF, with normal references to the associated JavaBean. Note there are some JSF tags we haven't seen before: the outermost tag is <f:subview>, since this is an included page; and h.outputLink, which includes an HTML tag in the results.
6. Open faces-managed-beans.xml in the WEB-INF subfolder. The <managed-bean> definition for the category page is the fourth one from the top.
7. There are no JSF-based navigation requirements for the category page (all navigation is handled within the page), but open faces-navigation.xml in the WEB-INF subfolder to see those navigations that are managed by JSF.

Integration with Hibernate:

The Spring framework is used to connect the view tier to Hibernate, which in turn interacts with MySQL. The process by which this happens, which is called "Inversion of Control" (IOC) is the subject for next session, but feel free to poke around in the Spring application context file (application-context.xml in the Web Pages -> WEB-INF folder) and see if you can get the connection (hint: start with the "sessionFactory" bean definition, then look for "<ref bean=" tags throughout the file). Note also that database connectivity is managed within the Spring application context file, so the JNDI lookups we defined earlier are not used. You might want to consider the pluses and minuses of this approach; it looks less clean on the surface, but what if we wanted to rip out Tomcat and replace it with a different application container (or, alternatively, rip out both Tomcat and JSF and run the application from the command line)?