Madura Perspectives Manager

User Guide

Document Revision: Revision: 1

Document Revision: Revision: 1

Table of Contents

1. Change Log

2. References

[1] MaduraObjects
[2] MaduraRules
[3] MaduraVaadinSupport
[4] MaduraBundles
[5] Vaadin
[6] Spring Framework
[7] Spring-Security
[8] madura-perspectives
[9] LogbackConfiguration

3. What is this?

I'll start with an example. Say you need an intranet application which presents your staff with various functions they need to do.

These functions might be:

- Recording and updating trouble tickets
- Entering their time sheets
- Looking at the current newsletter
- Viewing your product catalogue
- Maintaining customer details

We call these separate functions *perspectives*. They are delivered by sub-applications that are plugged into the main application, the manager. They can contribute various UI elements such as menu items and buttons. The manager decides which perspective's components should be visible at any time. This means the user can flick between various perspective-based applications while remaining in the same main (manager) application.

This has the following advantages over deploying the applications separately:

- Packaging the application as a Madura Bundle[4] is simpler than packaging it as a web application.
- The manager handles common functions such as login, bypassing the login if this has already been handled. The perspectives only need to assume they are already logged in.
- A blackboard system allows the different applications to communicate with each other if necessary, though they do most, or all, of their operations in isolation.
- New perspective applications can be added or removed dynamically to the manager without restarting the application environment. In flight updates of the perspective applications is supported. New logins see the newly added applications, otherwise the list of applications remains stable.
- Because there is only one CSS definition across all the application there is automatically a consistency in the look of all the perspectives.

The Manager uses Vaadin[5] and the perspective applications must use Vaadin to deliver their UI. There is no restriction on what parts of Vaadin the perspective applications can use.

There is more documentation about using Vaadin in this way in [3]

4. Running the Demo

The first thing you need to do is check out the project. See the Readme.md file in [8] for details.

This is a maven project with a default goal so make sure you have maven (v2) installed, type `mvn` and the war file will be built. You should be running Java 7 or later.

You will need an application server. We tested this with Tomcat V7, but any application server you are comfortable with should be just fine. It also runs on VMWare's cloud server, and probably most others.

For the purposes of keeping the demo setup simple the bundles it uses are embedded in the war file, that means you don't have to mess about telling your application server what directory to sweep to find them. The configuration to support the more flexible use is in the `applicationContext.xml` file, commented out, near the bottom. We will look at those options in *3*

To run the demo the user/password is admin/admin.

Figure (1) Madura Perspective Demo

What *Figure (1)* shows is the initial display when a user logs into the main application.

Worth noticing at this point is that there is not much there, but there is a list of sub applications on the left hand side. There is also a small menu and, over on the right, the 'Logged in as: admin' indicates who logged in.

So this is what the main application's UI looks like, but this is just a demo. The details of this UI can be customised to any requirements. For example if you want the list of sub-applications to appear somewhere else, perhaps over on the right, and as a drop down list you just change the code in the main layout. You probably do not want the Madura logo dominating the upper left hand side either, though you're welcome to put it there. Also you can adjust the CSS definitions that Vaadin uses to change colours and fonts etc. Vaadin themes are, of course, supported so you can select different themes for different users if you want. The selected theme is propagated to the sub-applications as well, ensuring the user has a consistent interface.

If the user selects the `User` option this loads the sub-application and changes the screen.

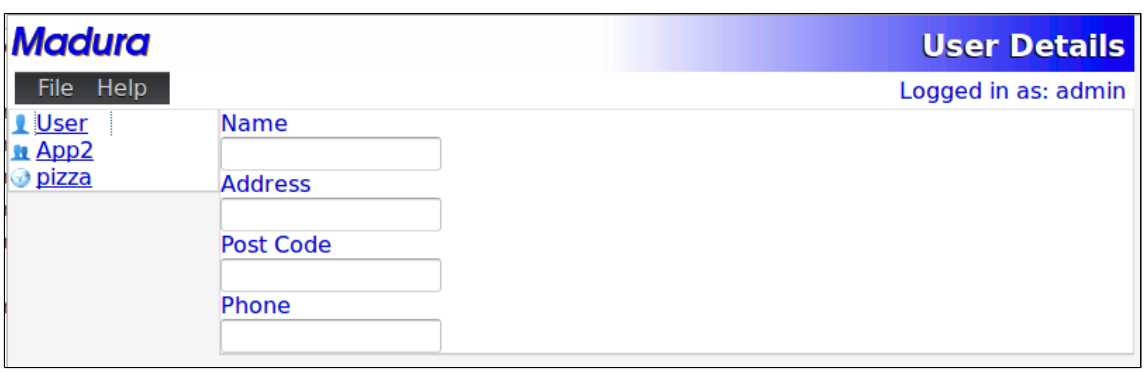

Figure (2) User Details

The User Details sub application is a very simple pure Vaadin form mapped to an object. It accepts information about a user but it does nothing much with it (this is only a demo). But it shows where the sub application UI appears on the screen. Notice that it retains the same theme as the main application. Notice the the title on the upper right has changed, it shows the name of the current sub application. If we now click the `pizza` option we see the `Pizza Order` UI.

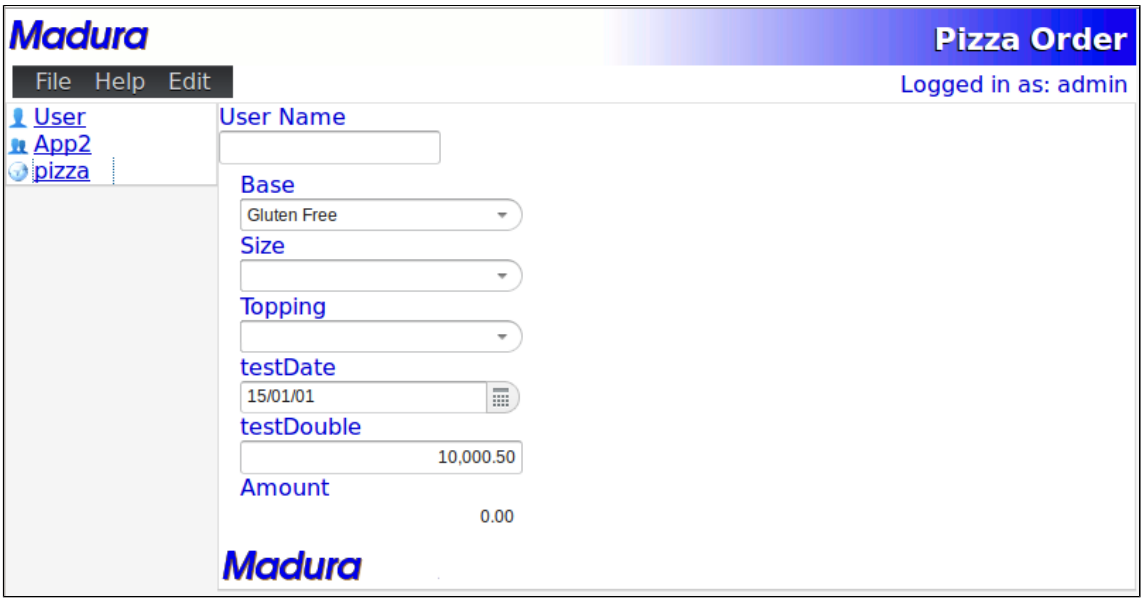

Figure (3) Pizza Order

This sub application uses Madura Objects and Madura Rules to drive the UI, so is a bit more interesting. It uses the same objects and rules as the pizza order demo. The point, though, is that this kind of functionality can be added to a sub application when we want. You might also notice that the menu has an extra entry. This was contributed by the Pizza Order sub application. Menu items appear and disappear as different sub applications are selected.

Now let's take a look at what these sub applications really are.

Each sub application is a Madura Bundle, which means it is a jar file with some extra bits, including a local Spring beans file. Each sub application contributes:

- A name. This is used to create the link on the left.
- A description. This is what you see displayed on the upper right. It is also used to make a tooltip on the link.
- A UI, which is the bulk of the application.

The sub applications can be added to the main application dynamically. Once a user has established a session, ie logged in, then they will not see the list of applications change, but they will get the latest list of available applications. This means you can dynamically add sub applications to the main application while it is still running. You can add new versions of an existing sub application in the same way. All you need to do is copy the jar file to the directory Madura Bundles is monitoring.

The applications use the same theme information as the main application, so a change of theme will automatically propagate to the sub applications.

Security information, specifically permissions established at login time, is also available across all sub applications.

And the sub applications can influence each other through a blackboard.

The blackboard is a publish-and-subscribe system where one application publishes something to the blackboard and other applications can react to the change or ignore it.

5. Configuring for Production

A production configuration would dispense with the bundles embedded in the war file and use an external directory. To configure this you need to tell your application server where that directory is.

```
<jee:jndi-lookup id="bundlesDir" jndi-name="java:/comp/env/BundlesDir"
 expected-type="java.lang.String" />
<bean id="bundleManager"
 class="nz.co.senanque.madura.bundle.BundleManagerImpl">
 <property name="directory" ref="bundlesDir"/>
 <property name="time" value="10000"/>
</bean>
```

You will find the above configuration in `applicationContext.xml` just above the bundleManager that is used by default, which you should remove, of course, because you only want one of them. The only difference between the two is that this one has the directory and time properties set. Just above it a JNDI name is defined. Now, you could simply hard code your directory in there and dispense with the JNDI name but in a production system you probably want to use JNDI so that you can change it without having to rebuild the application.

Also uncomment the last env-entry in the web.xml file.

The next step is to tell your application server what value to give that JNDI name. This depends on your application server. For Tomcat you can just edit it into your `context.xml` file like this:

```
<Environment name="BundlesDir" value="MY_DIRECTORY/bundles"
 type="java.lang.String" override="true"/>
```

Finally you want to actually add some bundles to that directory. There are three bundle projects you can use right away, these are the ones in the demo:

- madura-pizzaorder-bundle
- madura-demo-bundle1
- madura-demo-bundle2

These are all maven sub-projects and they are automatically added to the `WEB-INF/bundles` in the madura-perspectives-manager project. You can configure a sweep directory as described above and deploy your own bundle(s) to that. As long as you keep changing your bundle's version you can change, build and deploy it without restarting the main application, though you do have to log out and back in to see the new bundle you added.

A. License

The code specific to madura-perspectives is licensed under the Apache License 2.0 .

The dependent products have compatible licenses specified in their pom files. Madura Rules (optional) has a dual license to cover projects that do not qualify for the Apache License.

B. Release Notes

2.4.0

Implemented bundle scope (from new version of Madura Bundles)

2.3.1

Restructured the layout to include parent and sub-projects.

Added demo script link to menu.

Revised documentation.

2.3

Added internal bundles.

Added mechanism to specify the bundles dir in JNDI.

Fixed some I18n issues.

Switched to maven build.

2.2

Built for Java 1.7.

2.1

Updated for later Vaadin additions

2.0

Modifications to allow demo in Cloud Foundry

Initial release