**Zkfuse Demo Walkthrough** (version 0.1)

**Revision Log**

| **Ver. No.** | **Version Date** | **Summary of Change** | **Changed By** |
| --- | --- | --- | --- |
| 0.1 | 01/Oct/13 | Initial Version | Samuel Huang |
|  |  |  |  |

This document will attempt to walk the reader through the source code of **zkfuse** demo web application to provide an informal tutorial guide on how each demo is implemented, e.g. project layout, design patterns, tools used, database design, …

***A list of Demos in zkfuse web application***

1. ***Internationalization with Database***

This is the first and probably most complex demo implemented in zkfuse as it used a lot of tools (spring, JPA, hibernate, ZK, Mysql) and paves the way for project layout for the rest of demos.

1. ***Security Admin Console***

Prototype Security Admin Console for managing users and their associated roles and permissions. All these data will be persisted to MySQL.

1. ***Shiro Authentication and Authorization***

Demonstrate how easy it is to use [**Shiro framework**](http://shiro.apache.org/) to implement the login page. The logged in user will have privileges to enforce access of restricted page, depending on roles and permissions assigned to the user from demo 2 above.

1. ***JVM Information***

Display a simple page that displays the various memory settings (i.e. total, free, used and maximum) of JVM running the web application. *Every web app should have a backdoor like this to help troubleshoot performance issues*. ;)

1. ***Web Service Security Demo***

Showcase the use of WS-Policy (or [**WS-SecurityPolicy**](http://en.wikipedia.org/wiki/WS-SecurityPolicy)) within WSDL to implement Web Service Security by using the [**CXF framework**](http://cxf.apache.org/)

1. ***Encryption Demo***

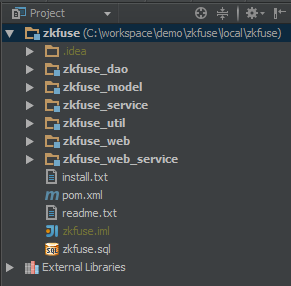
Show how easy it is to implement encryption and decryption of three algorithms: **AES**, **Twofish** and **Rijndael**. Of these three, AES (a variant of Rijndae) and Twofish (successor of Blowfish) are widely used today in real world.

***Project Layout***

Any decent Java developer will strive to structure their code in an Object Oriented way such that there is low coupling (degree to which a class knows about another class) and high cohesion (degree to which a class has single, well-defined purpose).

Now what does that mean for a Maven based web application? Break code into different layers of course! Each layer will correspond to a Maven module which has dependencies on other Maven modules or is depended on by other modules.

And since a picture is worth a thousand words, Figure 1 below sums up the maven project layout of zkfuse.



**Figure 1**: Project layout of zkfuse.

* **zkfuse\_model**

This is the model layer containing model/entity classes implemented using JPA. Each entity class would usually encapsulate the data of a table from database.

All the object-relational mapping is done in this layer using JPA annotations. Note cool kids will also use [Lombok](http://projectlombok.org/) to generate getter/setter, equals and toString methods of model classes here.

* **zkfuse\_dao**

This is the data access layer (or commonly called **DAO** layer) where DAO stands for data access object. This layer is implemented using Java Generics, Spring, Hibernate, JPA and JPQL.

For each entity class from zkfuse\_model module, there is always one corresponding interface and implementation class in this layer, e.g. say there exist an entity class Module in zkfuse\_model, then a corresponding ModuleDao interface and implementation class ModuleDaoImpl will exist in zkfuse\_dao.

Now classes in this layer will have the sole responsibility of access database directly, period. No classes of any layer should attempt to access database without delegating work to this layer.

This maven module has dependency on zkfuse\_model and zkfuse\_util.

Note [Spring Data](http://projects.spring.io/spring-data/) is said to be the successor of JPA with cool features that can simplify the implementation of DAO layer. It’s definitively worth looking into.

The properties to connect to database are set in **default.properties**. See install.txt for how to configure those. ***applicationContext-dao.xml*** defines all the spring beans used to access database with [declarative transactional support](http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/transaction.html). The service layer (i.e. zkfuse\_service module) will be able to use @Transactional annotation for declarative transaction because of the ***transactionManager*** bean declared here.

* **zkfuse\_dto\_service**

Note zkfuse\_dto\_service module doesn’t appear in Figure 1 above, what is it doing here? Well, an enterprise application really should have an extra DTO layer implemented as a maven module ***<project\_name>\_dto\_service*** to transfer data from JPA to a DTO (and vice versa). For simplicity, it’s not implemented used here.

This maven module has dependency on zkfuse\_dao.

*\** ***What is DTO (Data transfer object)?***

According to Wikipedia:

The difference between data transfer objects and [business objects](http://en.wikipedia.org/wiki/Business_object_%28computer_science%29) or [data access objects](http://en.wikipedia.org/wiki/Data_access_object) is that a DTO does not have any behaviour except for storage and retrieval of its own data ([accessors](http://en.wikipedia.org/wiki/Method_%28computer_science%29) and [mutators](http://en.wikipedia.org/wiki/Mutator_method)). DTOs are simple objects that should not contain any business logic that would require testing

\* ***Why do we need DTO layer?***

Sure, you will find blogs and articles saying DTO is evil, redundant (since entity object from model layer does the same job) and should be avoided but there are many valid reasons and scenarios for using it. Note for simple web app, the argumen*t*s against it are probably valid.

* + An entity/model object could have a large object graph (e.g. a Person object has multiple additional contacts, companies, Roles …, each entity object can in turn have many other entity objects) using JPA/Hibernate such that it’s impractical to use it for displaying data in view layer.

Such object could easily hold up to half MB of data in memory (sure you could have hibernate lazy loading but this can still happen if need to display data in object graph). It’s bad idea for server to hold this object in memory during life time of request and response or even session scope. If that doesn’t sound bad, then imagine 100 concurrent requests for different people. You get the idea.

By having a DTO layer, we could collect only the properties required for display in frontend by constructing a small DTO object and avoid aforementioned problem by setting the model object with large object graph to null right after the DTO is created. No need to wait for view layer to finish rendering the response to client.

* + This layer could assemble properties from multiple entity objects to create a DTO, which could then serve as the input argument to a business object (which contains business logic) in service layer.

Why? The service layer really shouldn’t be concerned with extracting properties from multiple entity objects just to create a business object with valid data. Simple data extraction logic could easily be refactored into DTO layer to provide clear separation of concerns.

Note if data extraction has business logic, don’t do it in DTO layer!

* + This layer also gives us the chance to massage and transform the data required for backend/frontend from one data type to another, e.g. frontend or web service may collect time as string but backend database needs to store it as data of type Time or Date.
  + The [Dozer](http://dozer.sourceforge.net/) and/or [Jakarta BeanUtils](http://commons.apache.org/proper/commons-beanutils/) frameworks can be used to [transfer data from entity object to DTO](http://blog.frankel.ch/tag/dozer) (and vice versa) automatically if both DTO and entity contain the same data types so the effort required is small. My current company is doing exactly this.
  + And lastly, it’s cool to have over-engineered and bloated solution to show how sophisticated it is. Ok, just kidding.

Note the DTO layer will access DAO layer directly to create DTO required for Service layer, nothing else. And this layer definitely has no business logic.

* **zkfuse\_service**

This is the service layer that will contain actual business logic and delegates call of accessing database to the DAO layer. The web layer (i.e. module zkfuse\_web) will use classes from this layer to get business logic done.

Transaction demarcation (i.e. marking boundary of database transaction) is also specified in this layer using Spring. For an excellent overview of how to properly use Spring annotations (e.g. @Transactional) to specify transactions, see [here](http://www.ibm.com/developerworks/java/library/j-ts1/index.html) and its related articles in the [series](http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=transaction+strategies).

Note in spring application context file applictionContext-service.xml, transactional support is enabled by the single line: *<tx:annotation-driven transaction-manager="transactionManager"/>*.

Note if DTO layer is used, then this layer should never use the DAO layer directly but use DTO layer instead to access database even if the DTO implementation is very simple. Reason being as soon as someone opens the door to use DAO layer directly, then other developers will join in and refactoring hell may soon follow.

This maven module has dependency on zkfuse\_dao if no DTO layer exists, else dependency on zkfuse\_dto\_service.

* **zkfuse\_util**

This maven module contains reusable utility classes that can be used by all other modules. Try not to have dependency on other maven modules here else circular dependency may happen.

* **zkfuse\_web**

This maven module is the web layer using **ZK** framework to render response page back to client. The controller will use both [MVC](http://books.zkoss.org/wiki/ZK%20Developer%27s%20Reference/MVC) and [MVVM](http://books.zkoss.org/wiki/ZK%20Developer%27s%20Reference/MVVM) design patterns to render response.

This module has dependencies on all other modules.

As for which Java web app framework is better for creating enterprise application (Uh-oh, getting into religious debate now)? After having used and played around with various frameworks ranging from Spring MVC, Seam, GWT, SmartGWT, ..., I have to say ZK beats them all.

Proof? See demos in this [page](http://www.zkoss.org/zkdemo/getting_started), [ZK vs. GWT : Server-Centric Matters](http://ria.dzone.com/articles/zk-vs-gwt-server-centric-matte-1), [ZK vs GWT](http://www.zkoss.org/whyzk/zk_vs_gwt), and [this](http://jawsy.fi/blog/2011/09/08/thoughts-about-the-zk-web-framework-overall-experience/). You just don’t get to pack so much Ajax behaviour and nice UI widgets with so little code in other frameworks. This is not to say I will have problem using a different framework in my next job. And no, I am not affiliated with ZK. ;)

For people who can’t live without **jQuery UI** for frontend, ZK actually supports using [jQuery UI widgets](http://books.zkoss.org/wiki/Small_Talks/2010/December/How_to_Wrap_a_jQuery_UI_Widget). I haven’t tried this before so I can’t comment how cool it is.

At end of day, I guess there is no such thing as the best framework, it all comes down to choosing the right tools for the right job given the skill set of people available in team with the resource available to get job done in minimal time without sacrificing the quality of work.

* **zkfuse\_web\_service**

This maven module contains all the web service implementation. See *Web Service Security Demo* for what web services are available.

***Demo 1 – Internationalization with Database***

**Objective:**

To internationalize labels of web application by database

**Motivation:**

The traditional way of doing internationalization in JEE application is by storing labels in properties files. One properties file for the locale (i.e. language) of each targeted page/module.

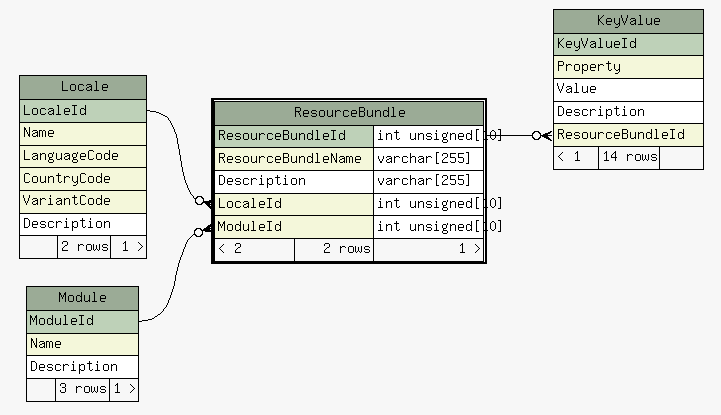
This all sounds fine until you want to change a label in dev or production. To achieve this, one has to change the label in a properties file then restart the server or redeploy the web application for change to take effect.

Now imagine the application is to be internationalized for a new language, we have to repeat the above step for a set of labels within a page, test it then repeat it again for the next set of labels. This is both tedious and time consuming, not to say if you need to change a label on the fly for production simply because the clients don’t like it!

**Solution:**

What if we were to do internationalization through admin GUI in browser and persists all labels in database? Then all the labels could be updated or created on the fly without redeploying the application.

**Database Design:**

****

**Figure 2:** Database design of Internationalization Demo. Generated by SchemaSpy.

**Solution Design:**

Whenever a page is requested, all labels of that page will be retrieved as KeyValue records having the same parent ResourceBundle record referred to by the foreign key ResourceBundleId of KeyValue. This ResourceBundle record is retrieved by using both Locale record (current locale from session) and Module record (each page represents a module having a unique module name set in controller). See Figure 2 above.

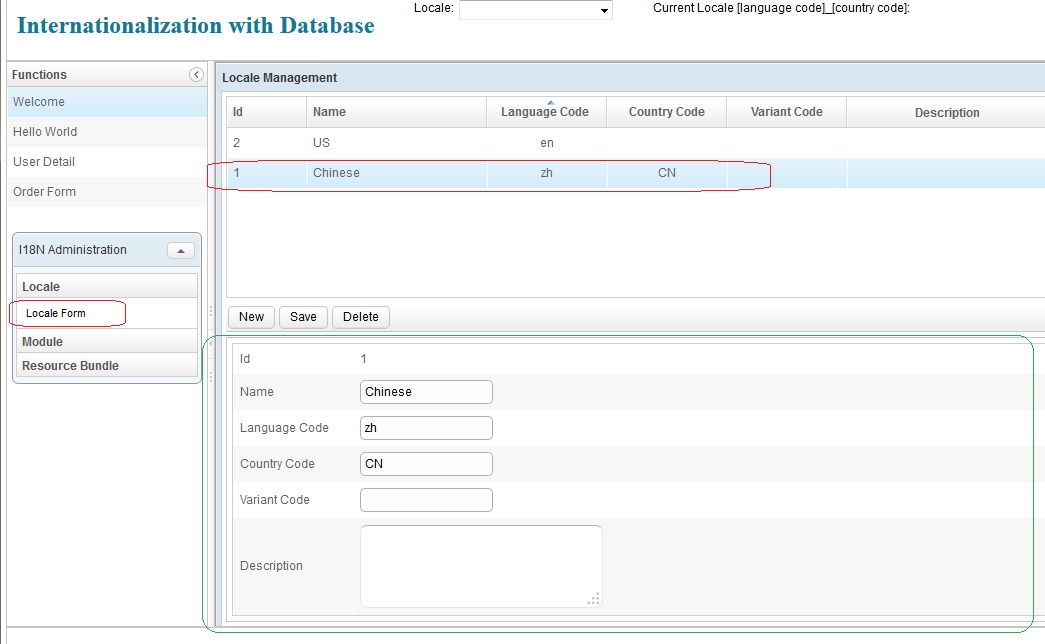
In the ER diagram of Figure 2, the crow-feet end of relationship lines represent zero or more child records so both Locale and Module could have multiple (or zero) child ResourceBundle records. LocaleId and ModuleId fields of ResourceBundle are the foreign keys referring to the primary key of Locale and Module respectively.

To prevent accessing the database repeatedly to retrieve KeyValue records for the same page, all the retrieved KeyValue records are cached to serve future requests of the same page to optimize performance.

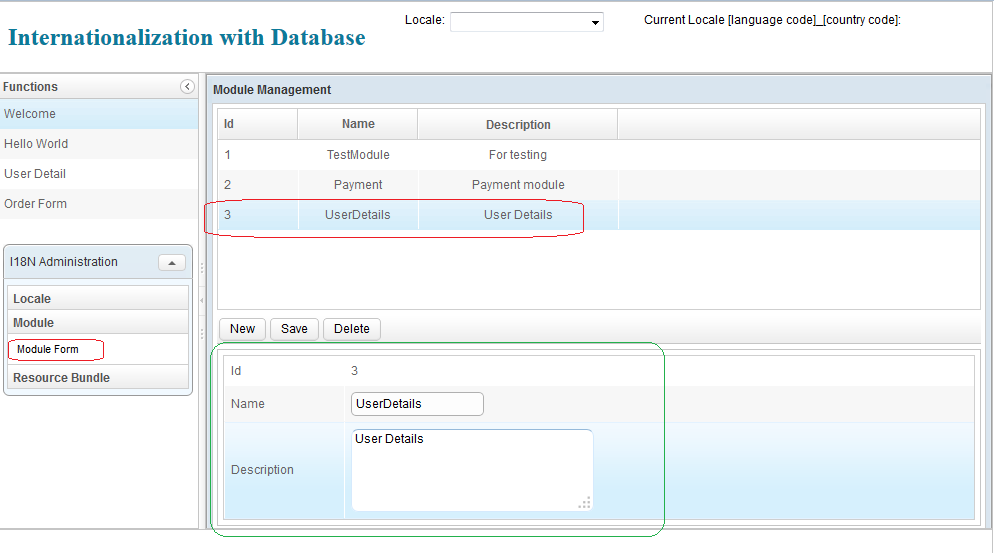
This does leave us the questions of how to manage Locale, Module, ResourceBundle and KeyValue records dynamically at runtime so no server restart or application redeploy is required. Hence the next section *Frontend Design*.

**Frontend Design:**

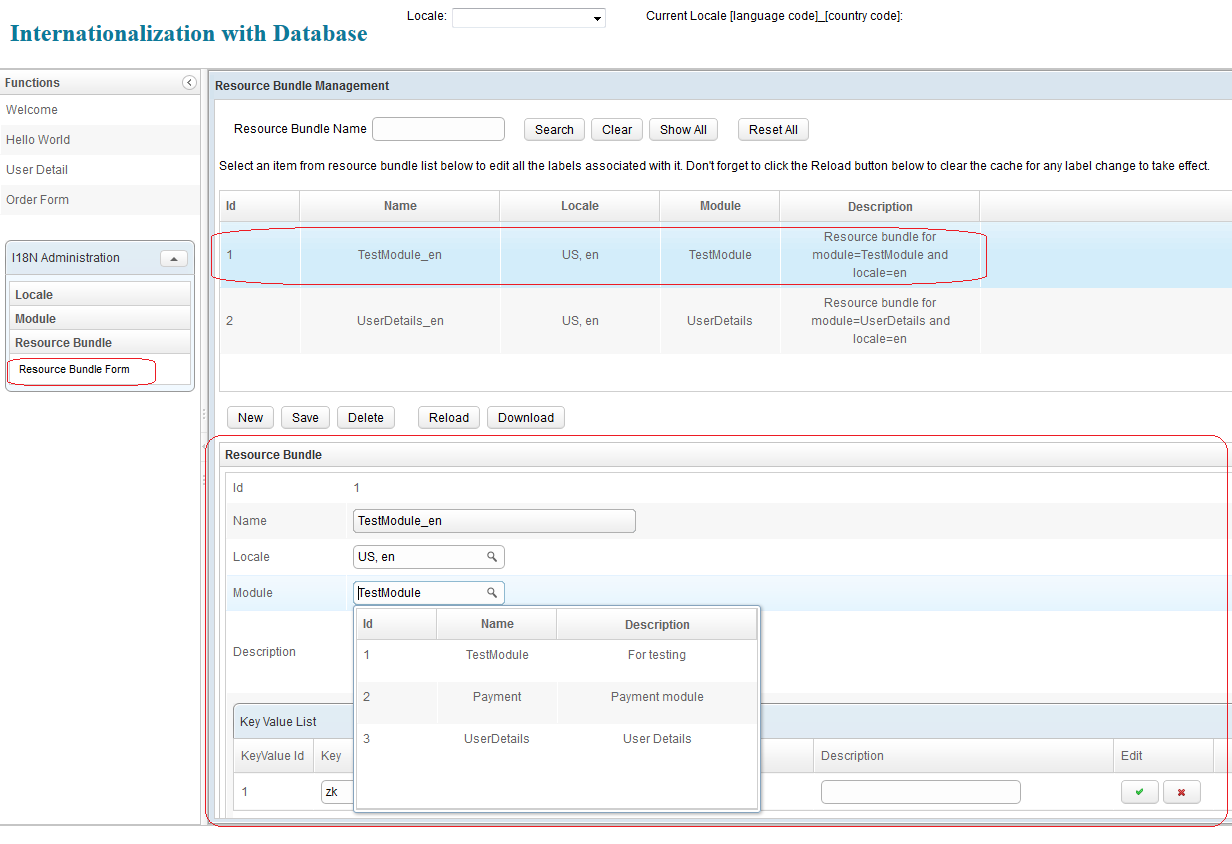
To answer the question of managing records for tables in Figure 2 at runtime, an Admin Console will be required. Let’s call this *I18N Admin Console*. This console will need to have at least four panels, one for each table.



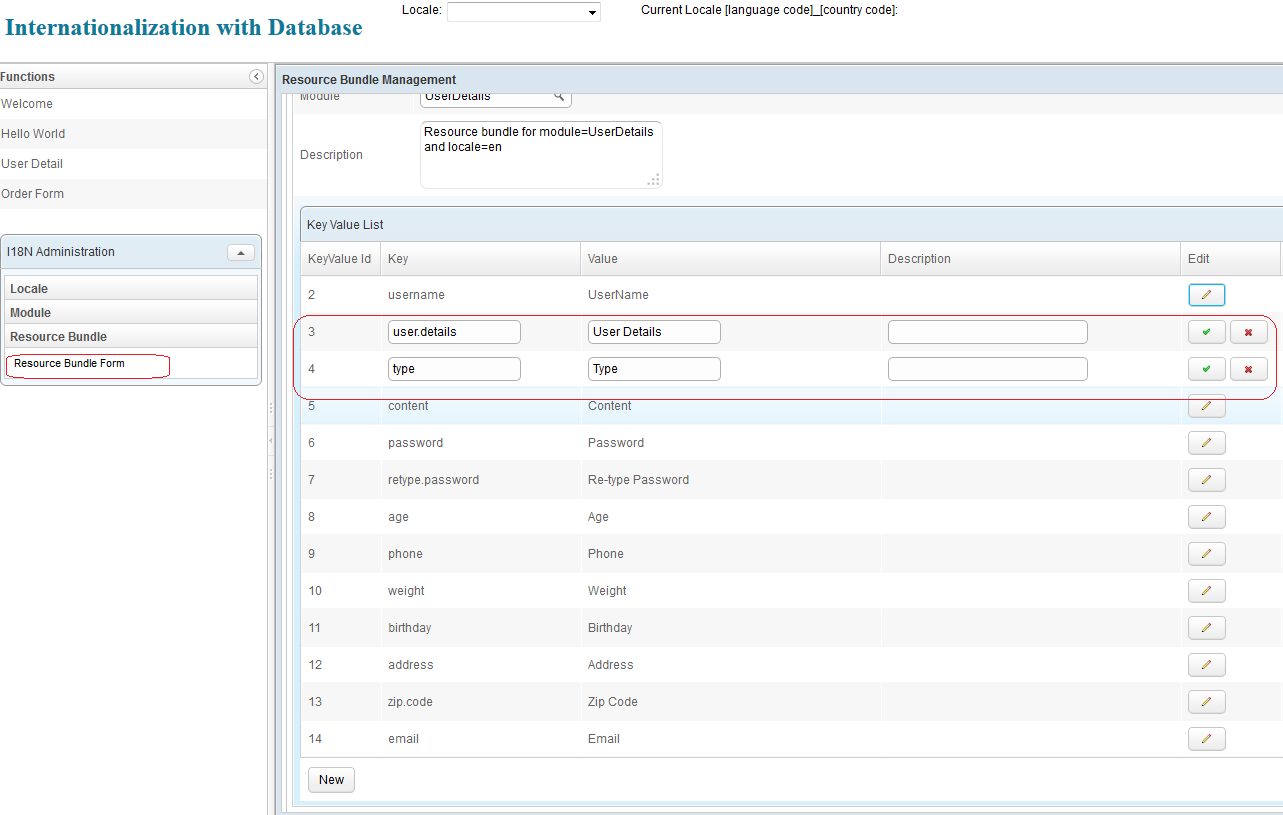
**Figure 3**: Locale management panel. The page content highlighted in green rectangle will change via Ajax depending on which locale is selected (highlighted red rectangle) from the locale table.



**Figure 4:** Module management panel. The page content highlighted in green rectangle will change via Ajax depending on which module is selected (highlighted red rectangle) from the module table.

**Figure 5:** Resource Bundle management panel. The page content highlighted in red rectangle at bottom will change via ajax depending on which resource bundle is selected (highlighted red rectangle at top) from the Resource Bundle table.

Now if you look closely at figure 5, you can see ‘Key Value List’ panel is embedded within Resource Bundle management panel. This is because a resource bundle has a set of key values (i.e. labels) so wrapping key value panel here should make the GUI more intuitive to use.



**Figure 6:** Key Value panel. To edit a label, click the Pencil button. To save label change after edit, click the Tick button. To delete a label, clicks the X button.

For the label changes to take effect, click the Reload button in Figure 5 to clear the cache of that resource bundle. This button will only be enabled when a Resource Bundle is selected from the list/table in Resource Bundle form.

**Code Review:**

Since there are too many things going on here, I will only review on a few things I think is cool or important and leave the rest for readers to find out by examining the source code.

* ***CRUD operations of each panel***

[MVVM](http://books.zkoss.org/wiki/ZK%20Developer%27s%20Reference/MVVM) (Model-View-ViewMode) design patterns from ZK framework has been chosen to implement this rather than the traditional MVC framework. Why? Because MVVM in ZK has top notch data binding support that make implementing CRUD operations a breeze.

The way it works is, you define a ViewModel class (for each entity class) containing the data model of entity objects, e.g.

private ListModelList< {entity object} > **entityList** = new ListModelList< {entity object} >();

The ViewModel class will then use Service class from Service layer to populate this ***entityList*** data model when certain event gets triggered.

Now in the frontend zul file (equivalent of JSP in ZK), we can declare the use of this ViewModel class in a Div, Window or Panel UI component. Within that UI component, we then bind Listbox component (which generates a table/list with one entity record per row) to the data model.

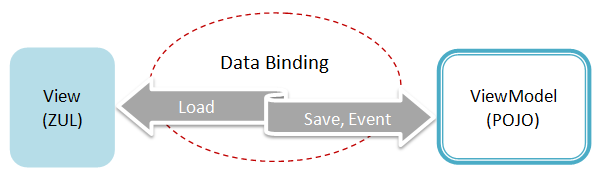
We can then bind a set of UI components, say text boxes, combo boxes, … to display properties of selected entity instance in the listbox via ajax, e.g. the highlighted green rectangles area get updated via ajax whenever a selection is made in listbox above in Figures 3 and 4 above.

No big deal right? But now the data between the view (zul file) and the ViewModel is synchronized so that any change made to the data in UI components is automatically carried over to the target ViewModel based on a predefined binding relationship (and vice versa).

Say, for example the user clicks on a button in ZUL to trigger some business logic that adds/removes data from the entityList of ViewModel. Once that is done, all UI components displaying data from the data model *entityList* should be updated as well manually in ViewModel or whatever class to reflect the changes in the View right?

Nah, the [**Data Binding**](http://books.zkoss.org/wiki/ZK%20Developer%27s%20Reference/MVVM/Data%20Binding) feature of ZK will take care of this so all UI components will update itself via ajax to show new data automatically. And there are many different ways to bind. See this [link](http://books.zkoss.org/wiki/ZK%20Developer%27s%20Reference/MVVM/Data%20Binding) for more details. This is data binding from ViewModel to View (ZUL file).

What about data binding from View to ViewModel class? Well, there are a bunch of annotations that can be specified as attributes on UI components of ZUL file to bind the component’s data, UI events (e.g. onClick, onChange) and command (i.e. the name of event listener registered on any ViewModels) along with parameters to … push data from View to ViewModel. This in turn can trigger some business logic to change the data model entityList then all data changes can get pushed back to View via ajax automatically as above.



**Figure 7:** Data binding between View (ZUL file) and ViewModel class

If this still doesn’t sound much then try spicing things up with multiple data models from multiple ViewModels, all used within a single page, updating each other’s data (do this only if you need to, just because you can doesn’t mean you should).

In case you are wandering, a method in ViewModel class can be converted to an event handler to listen for UI events fired from the View by just adding [@Command](http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/MVVM/Data_Binding/Command_Binding) annotation and fire events to update UI components that’s bound to some data (i.e. instance variables) within ViewModel with another [@NotifyChange](http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/MVVM/ViewModel/Notification) annotation, e.g.

*private E selected; // the selected item of listbox in View, properties of this entity object are bound to UI components*

*ListModelList<Locale> localeList; // this is bound to listbox in View*

***@Command****(“okidoki”) // listen UI events with command okidoki*

***@NotifyChange****({"****selected****","****localeList****"}) // update whatever UI componets bound to these properties via ajax*

*public void newEntity() {*

*newLocale();*

*}*

What does the data binding code look like in frontend? Well,

<window title="Locale Management" border="normal" width="100%" contentStyle="overflow:auto"

apply="org.zkoss.bind.BindComposer" **viewModel="@id('vm') @init('org.zkfuse.web.vm.i18n.LocaleVM')"**

validationMessages="@id('vmsgs')">

<vbox hflex="true">

<listbox **model="@load(vm.entityList)"** **selectedItem="@bind(vm.selected)**" hflex="true" height="200px">

<listhead>

…

</listhead>

<template name="model" var="item">

<listitem >

…

</listitem>

</template>

</listbox>

<toolbar>

<button id="newBtn" label="New" onClick="**@command('newEntity')**" autodisable="newBtn,saveBtn,deleteBtn" />

<button id="saveBtn" label="Save" onClick="**@command('saveEntity')**" disabled="**@bind(empty vm.selected)**" autodisable="newBtn,saveBtn,deleteBtn" />

<button id="deleteBtn" label="Delete" onClick="**@command(empty vm.selected.localeId?'deleteEntity':'confirmDelete')"** autodisable="newBtn,saveBtn,deleteBtn"

disabled="**@load(empty vm.selected)**" />

</toolbar>

<groupbox form="**@id('fx') @load(vm.selected) @save(vm.selected, before='saveEntity')"** visible=**"@bind(not empty vm.selected)**" hflex="true" mold="3d">

<grid hflex="true" >

<columns>

<column width="120px"/>

<column/>

</columns>

<rows>

<row>

Id

<hlayout>

<label value="**@bind(fx.localeId)**" />

<image src="**@load(fxStatus.dirty?'exclamation.png':'')"** />

</hlayout>

</row>

<row>

Name

<hlayout>

<textbox id="namebox" value="**@bind(fx.name)** @validator(vm.jsr303Validator)" mold="rounded" />

<label value="**@load(vmsgs[namebox])**" sclass="red" />

</hlayout>

</row>

<row>Language Code

<hlayout>

<textbox id="langbox" value="**@bind(fx.languageCode)** @validator(vm.jsr303Validator)" mold="rounded" />

<label value="**@load(vmsgs[langbox])"** sclass="red" />

</hlayout>

</row>

<row>Country Code

<hlayout>

<textbox id="countrybox" value="**@bind(fx.countryCode)** @validator(vm.jsr303Validator)" mold="rounded"/>

<label value="**@load(vmsgs[countrybox])**" sclass="red" />

</hlayout> </row>

<row>Variant Code <textbox value="**@bind(fx.variantCode)**" mold="rounded"/></row>

<row>Description <textbox value="**@bind(fx.description)"** rows="4" cols="40" mold="rounded"/></row>

</rows>

</grid>

</groupbox>

</vbox>

**Listing 1:** Code snippet from localeForm.zul

As one can see, it’s not really difficult. The actual annotations (highlighted in bold) that do data binding above are @bind, @load and @save. Events are fired with @command annotation. The sample code will render ‘Locale Management’ panel in Figure 3. The whole code is available in *localeForm.zul* from source code.

Data binding that allows data to go both ways is called two way data binding. It’s achieved with **@bind** annotation as in sample code above while **@load** does one way binding from ViewModel to View. Lastly **@save** and **@command** annotations do one way data binding from View to ViewModel. Confused? See Figure 7.

In the demo, since all ViewModel classes have CRUD operations, I created an abstract GenericVM base class declaring command methods (method annotated with @Command) for all CRUD operations. That is, newEntity(), getEntityList(), deleteEntity() and saveEntity(). Then it’s then up to the subclass to provide real implementations using service class.

Note in MVVM design pattern, there is no use of controller to render View. Just ZUL file and ViewModel class. It is possible to use ViewModel within controller so we have the best of both design patterns for more complicated stuff but it’s out of the scope of this walkthrough. This [thread](http://forum.zkoss.org/question/76567/mvvm-and-mvc-mix-communication-createcomponents/) provides one way of doing it.

**Gotchas** of using MVVM design pattern? Well, the major one can be summed up in the following paragraph quoted from this [link](http://books.zkoss.org/wiki/ZK%20Developer%27s%20Reference/MVVM/Data%20Binding)

But *ViewModel should contain no reference to UI components* and knows nothing about View's visual elements. Hence there is a clear separation between View and ViewModel, this is also the key characteristics of MVVM pattern.

Now this is definitively not possible when UI is complex with business logic so the use of both MVC and MVVM to render a page will be the best practice. Note there is nothing to stop you from putting UI components in ViewModel with business logic manipulating those components. Do it only as your last resort for production code.

More Reference

* [Contrast MVVM in ZK 6 with MVC](http://books.zkoss.org/wiki/Small_Talks/2011/December/MVVM_in_ZK6:in_Contrast_to_MVC)
* ***Caching Labels of each page***

So what’s the sequence of events for retrieving labels from database? Good question. Even I am confused (seriously) even though I wrote and designed this stuff so I will document it below.

Whenever a label is requested in a page (i.e. LabelUtils.getLabel(key)), the current locale and module name will be retrieved from the current ZK session to serve as inputs to retrieve a *DBResourceBundleLocator* object from a static ConcurrentHashMap (since it provides concurrent access to multiple threads and we want it to be thread safe) of <Locale, Map(String moduleName, DBResourceBundleLocator dbResourceBundleLocator)> in LabelsUtil class.

This retrieved *DBResourceBundleLocator* object will in turn contain a ConcurrentHashMap of <String labelKey, String labelValue> which is used to retrieve the actual label, e.g. map.get( labelKey ). This map will contain a set of key value pairs populated from the KeyValue records.

*LabelsUtil has*

private static Map<Locale, Map<String, DBResourceBundleLocator>> **dbResourceBundleLocatorMap** = new ConcurrentHashMap<Locale, Map<String, **DBResourceBundleLocator**>>();

*DBResourceBundleLocator has*

private Map<String, String> **keyValueMap** = new ConcurrentHashMap<String, **String**>();

The actual label

**Listing 2**

Why all the trouble of this mumbo jumbo? So we never have to access database twice to retrieve labels of a given locale for a page. You see, whenever a page is accessed, a method (i.e. *BaseBindComposer. registerNonExistingDBResourceBundleLocator ()*) will check to see if DBResourceBundleLocatorobject exist within the first ConcurrentHashMap of LabelsUtil for the given locale and module name.

If it exists already, do nothing. If it doesn’t exist, then we register it in the map by:

1. CreatingaDBResourceBundleLocator object
2. Populate the map *keyValueMap* of DBResourceBundleLocator object by retrieving ResourceBundle object with matching locale and module name from database. This ResourceBundle object will contain a list of KeyValue objects that we can then use to populate the map (see listing 2).
3. Register the DBResourceBundleLocator object to the static map *dbResourceBundleLocatorMap*

The end result is a label will be cached and retrieved from the static map **dbResourceBundleLocatorMap** of LabelsUtil. No need to access database again after first access. Performance improvement, Yay!

One more thing, each page will need a unique module name. Where is this module name set exactly? Well, we hard code it as a static string variable in ViewModel class (e.g. UserDetailsVM) for MVVM or in controller for MVC design pattern. Reader may frown upon hearing hard coding it but hey, if you need to change the module name of a page or panel dynamically at runtime, then something is seriously wrong with the design.

***Demo 2 – Security Admin Console***

**Objective:**

To prototype Security Admin Console for managing roles and permissions associated with a user

**Motivation:**

Any web application that will allow a user to log in to access different pages and functions depending on user’s privileges will need to manage user’s privileges as a set of roles and permissions. Hence the need for Security Admin Console.

Such Admin Console is also required for managing *Service Account*. What’s a Service Account? Basically, multiple users of an organisation/team could use the same service account to access a service (e.g. web service). So by changing the roles and permissions attached to the service account, all users’ access can be changed at once rather than changing access of all users one by one.

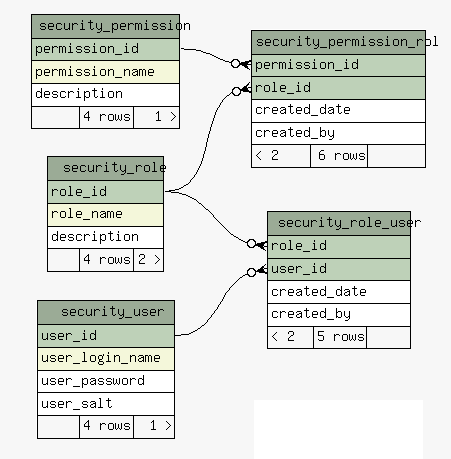
**Database Design:**

We basically have three domain objects to consider here: User, Role and Permission.

A User (or service account) can have a set of Roles assigned to it. Each Role can also have a set of Permissions assigned to it for fine-grained authorization.

Now since each role can be assigned to multiple users, we have many-to-many relationship between User and Role. And since permission can also be assigned to multiple roles, we have another many-to-many relationship between permission and role.

Now good database design dictates that we break up the many-to-many relationship into two one-to-many relationships using an association table containing at least the primary keys of both original tables. So what we end up with is:

****

**Figure 8:** Database design of Security Admin Console demo

**Solution Design:**

The magic of all relationship mappings happen of course in the entity classes of model layer using JPA annotations. The package containing entity classes for this demo is org.zkfuse.model.security in zkfuse\_model module. I won’t spend time explaining how the mappings are done here since there are plenty of excellent online tutorials explaining those quite well. And frankly they are what I based my implementations on. You can find the tutorials [here](http://uaihebert.com/?p=1674&page=22), [here](http://giannigar.wordpress.com/2009/09/04/mapping-a-many-to-many-join-table-with-extra-column-using-jpa/) and [here](http://www.mkyong.com/hibernate/hibernate-many-to-many-example-join-table-extra-column-annotation/). Each table from Figure 8 will have its own entity classes.

Along the journey of JPA/Hibernate mappings, I run into many confusing concepts that got cleared up in stackoverflow. To save others the pain of going through those, I will document some links here: [the inverse side of the association](http://stackoverflow.com/questions/2584521/in-a-bidirectional-jpa-onetomany-manytoone-association-what-is-meant-by-the-in), [JPA JoinColumns vs mappedBy](http://stackoverflow.com/questions/11938253/jpa-joincolumn-vs-mappedby).

**Testing:**

Unit tests in the DAO layer of zkfuse\_dao module is an absolutely must to confirm the JPA relationship mappings in the model layer is working as expected. This is also where most of the problems can get ironed out ASAP.

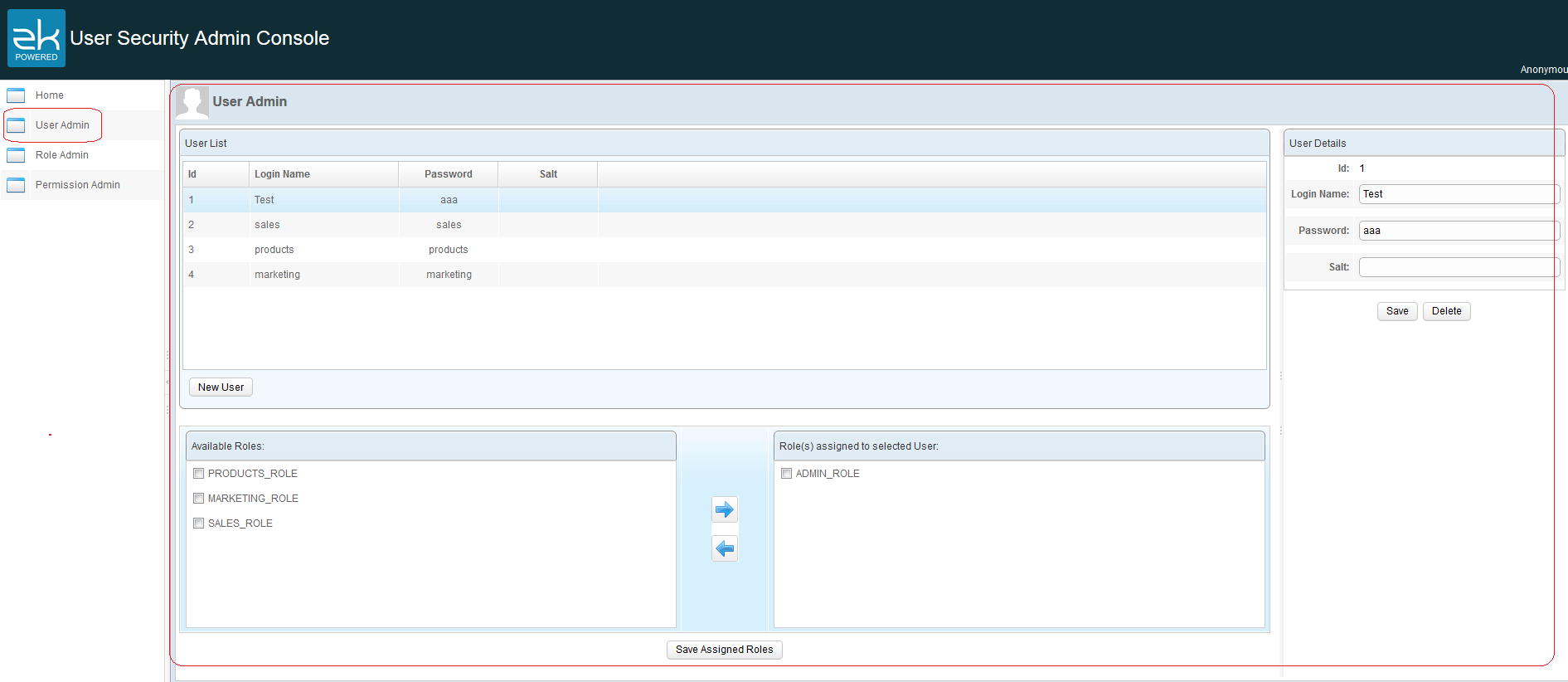
And no, writing excessive unit test is not waste of time. Or at least that’s what you need to believe if you call yourself a TDD practitioner. Note the inverse of TDD is DDT, which is used to kill bugs!

The package for testing persistence of entities for this demo is in org.zkfuse.dao.security.impl of zkfuse\_dao module. Just make sure the unit tests are repeatable (no dependency on existing data in database), short and concise, test one thing per testing method and DO IT UPFRONT before using the DAO classes in service and web layers.

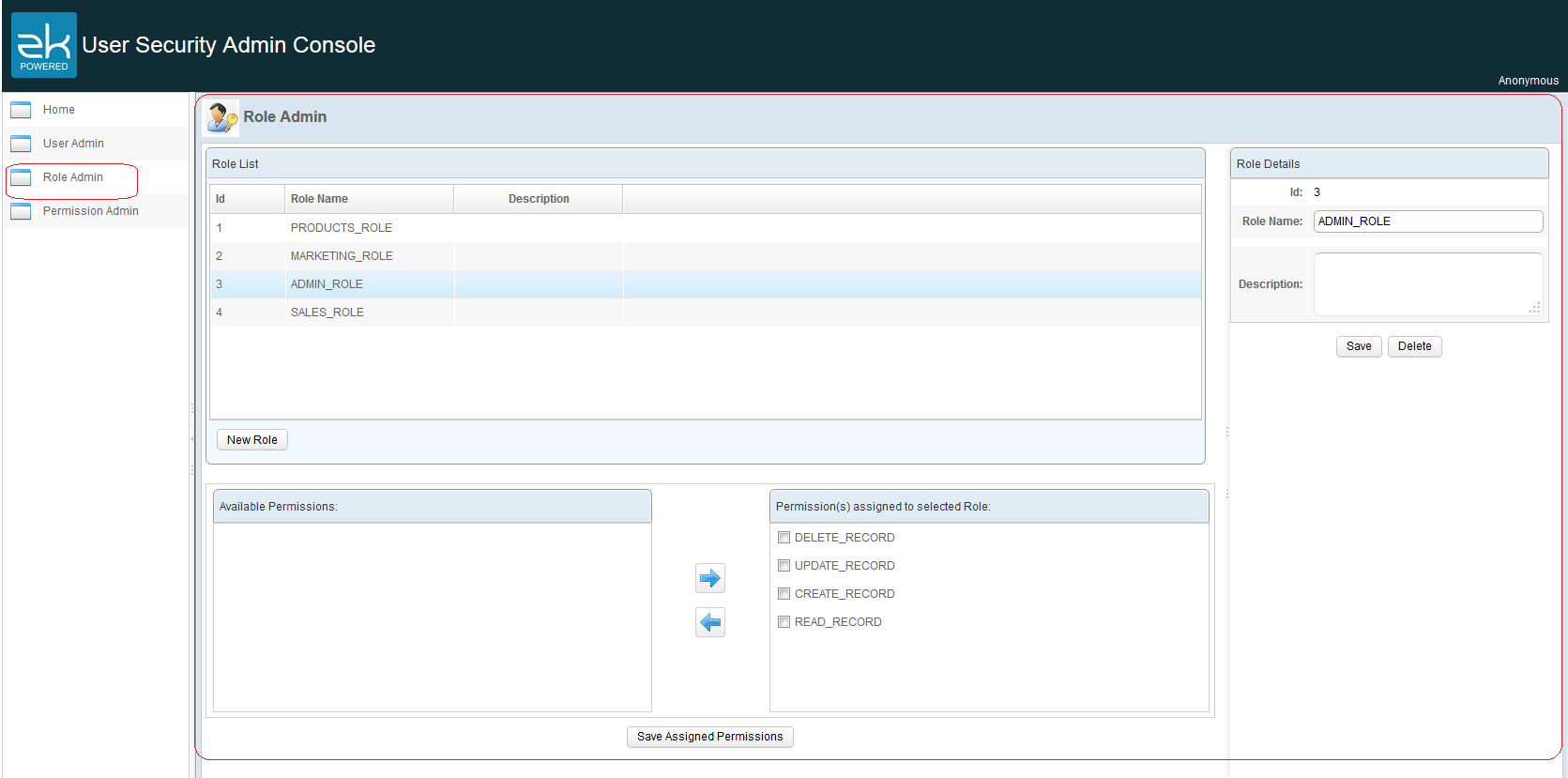
**Frontend Design:**

Three panels are required. One panel for managing user details and assigns roles to the selected user. A second panel will need to manage roles and assign permissions to the selected role. The last panel will manage permissions. Needless to say, the usual CRUD operations will need to happen in all three panels for each entity as well.

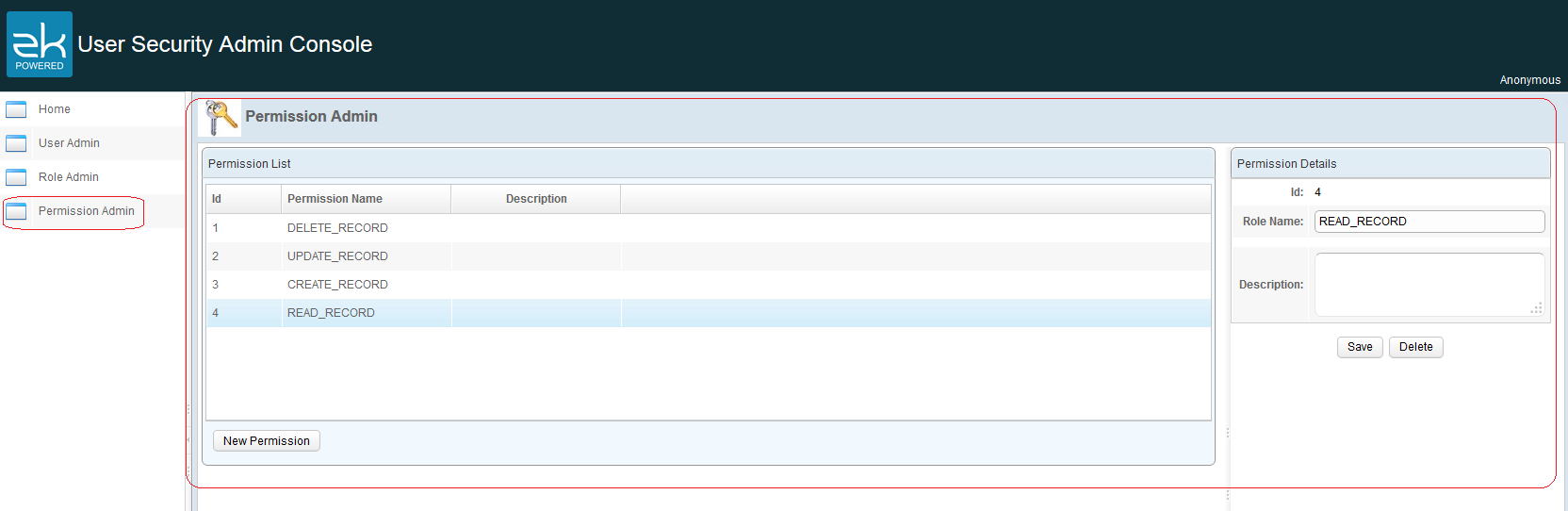
As a bonus, everything should happen via Ajax without reloading the page. Can we do it? By the power of ZK, yes we can!



**Figure 9:** User Admin panel for managing user details and assigning roles to the selected user.



**Figure 10:** RoleAdmin panel for managing role details and assigning permissions to the selected role.



**Figure 11:** PermissionAdmin panel for managing permission details.

**Code Review:**

What can I write to fill up the rest of this page? I think I will explain how the ajax-based navigation feature is implemented here to start with.

In figures 9, 10 and 11, the content area highlighted by large red rectangle will change via ajax depending on what link is clicked on the left hand navigation bar. This is achieved through a combination of **templating** and **ajax-based navigation**. Note the templating mechanism is very similar to SiteMesh for those who have used SiteMesh before.

Then I remembered there is an excellent tutorial in [ZK Essentials](http://books.zkoss.org/wiki/ZK%20Essentials) series that talked about this already in great details. So I will simply redirect your attention to [this chapter](http://books.zkoss.org/wiki/ZK%20Essentials/Chapter%207:%20Navigation%20and%20Templating). Job done! My implementation of ajax-based navigation is actually based on the downloadable source code of ZK Essentials.

What about the cool feature of assigning roles to user (Figure 9) and assigning permissions to role (Figure 10)? How difficult is that? Unfortunately, not difficult at all. My implementation is based on in this [Column Layout demo](http://www.zkoss.org/zkdemo/zk_pe_and_ee/layout_column_layout) after stripping out commercial components. The exact files that implement this feature in my demo are user-admin.zul and role-admin.zul.

For those wandering how ajax-based data binding for CRUD operations is done here, please refer to the ***CRUD operations of each panel*** section above.

The same template mechanism and ajax-based navigation used here will also be used in *Shiro Authentication and Authorization Demo* so I won’t mention these in the Shiro demo latter.

***Demo 3 – Shiro Authentication and Authorization***

**Objective:**

Implement a login screen using the Apache Shiro framework. After logged in, user should only be able to access a page if he/she has been assigned the role to access that page.

**Motivation:**

This is probably the most basic and must have feature of any enterprise web application. That is, being able to login to a web application and access services depending on the privileges (i.e. roles and permissions) assigned to your login account.

Why use [Apache Shiro framework](http://shiro.apache.org/)? Why not use Spring Security 3 framework, which is arguably the de facto security framework in JEE world?

Well, … are you crazy? No offense to any Spring Security hot shot. Spring Security 3 is way … more difficult to use and learn than the Shiro framework. I actually have the book [Spring Security 3](http://www.packtpub.com/spring-security-3/book) so I do know what I am talking about.

Proof? Well, see [Shiro vs. SpringSecurity](http://stackoverflow.com/questions/4991084/shiro-vs-springsecurity), [10 Minute Tutorial on Apache Shiro](http://shiro.apache.org/10-minute-tutorial.html) and [Application Security With Apache Shiro](http://www.infoq.com/articles/apache-shiro).

**Database Design:**

The same database used in Security Admin Console Demo above will be used here.

**Solution Design:**

The solution basically comes down to both authentication and authorization.

Authentication is achieved by logging in. This will be implemented as the classic username/password login form. To login successfully, both username and password from the form need to match the user\_login\_name and user\_password fields of security\_user table respectively in figure 8.

Authorization can be implemented by matching user’s roles against the role required to access a page. If one of the user’s roles matches the page role, then the user is granted access to the page, else an access denied message should be displayed to the user.

The user’s role will be mapped to the role\_name field of security\_role table in Figure 8. After successful login, the user must also be able to log out else a potential security hole. And yes, a user can have many roles.

For this demo, three pages will be created to prototype the solution. Each page will require a different role to access. They are:

\* Marketing Admin Console: requires logged in user with '*MARKETING\_ROLE*' Role to access

\* Sales Admin Console: requires logged in user with *'SALES\_ROLE*' Role to access

\* Products Admin Console: requires logged in user with '*PRODUCTS\_ROLE*' Role to access

Note a logged in user with Admin role ‘ADMIN\_ROLE’ should be able to access all pages. Remember each role can have multiple permissions assigned to it for fine-grained authorization in Security Admin Console demo? No permission (i.e. security\_permission table) will be used here to decide what the user can access for simplicity.

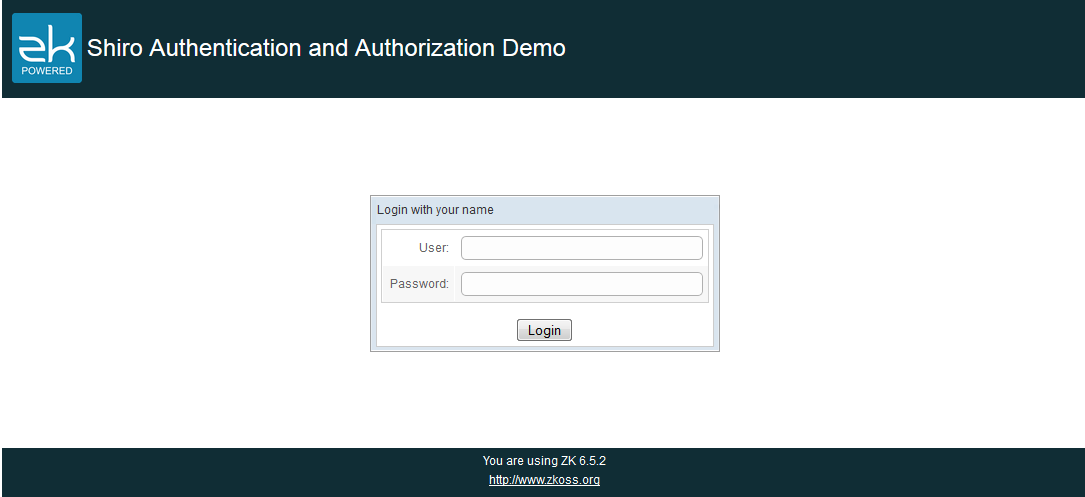
**Testing:**

To create a new user with the required role to access the target page, use the Security Admin Console from previous demo. Once done, log in as that user and see if the user can access the target page. Log out.

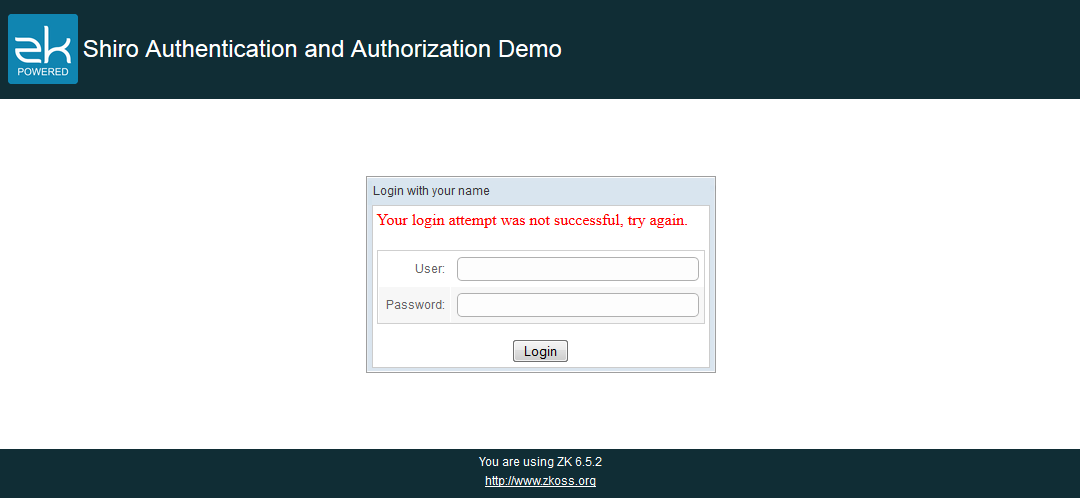
Then remove the user’s role through Security Admin Console. Try login again and see if user can access the target page again. An access denied error message should return.

**Frontend Design:**

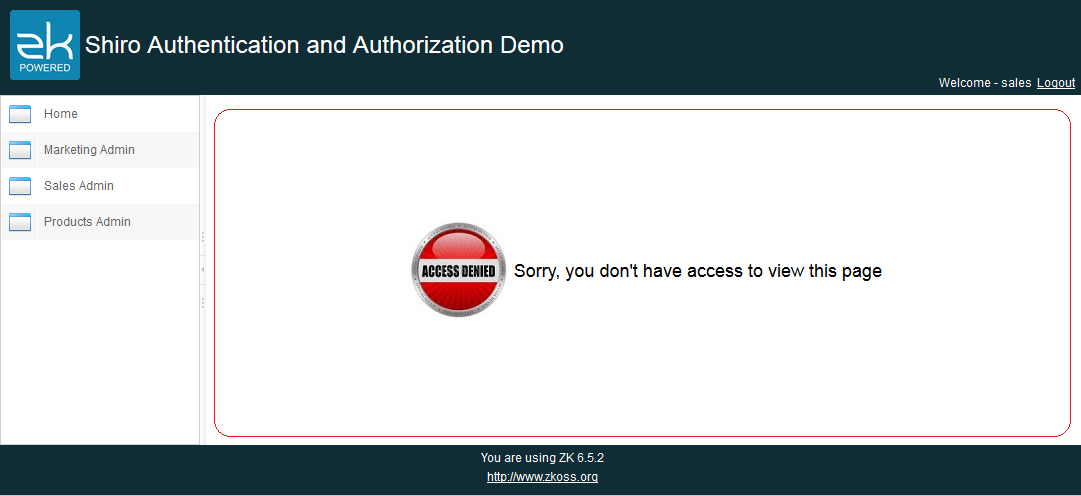
The following figures summed up the design of all GUI.

****

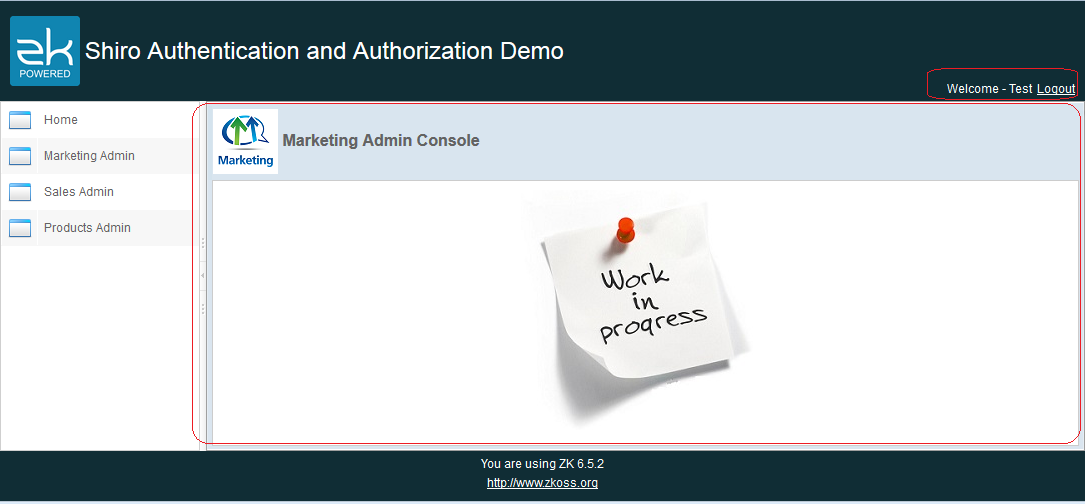
**Figure 12:** Login Form

****

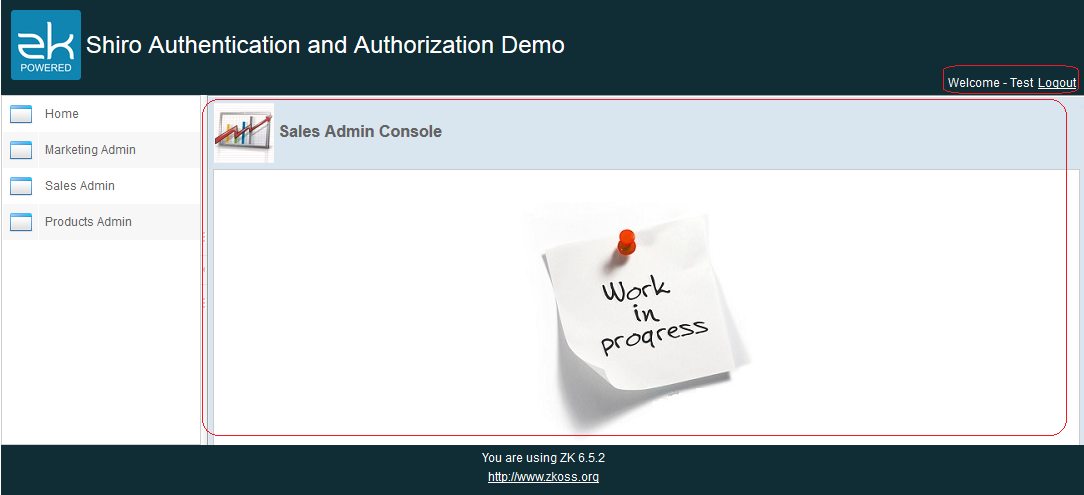
**Figure 13:** Login Form with error message after failed login

****

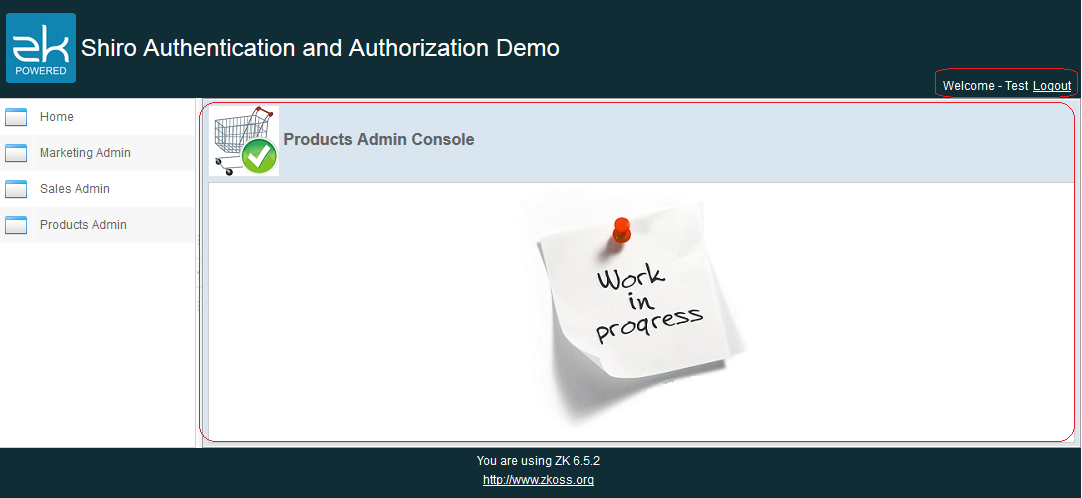
**Figure 14:** A reusable access denied panel that will show when the user tries to access a page without the required role



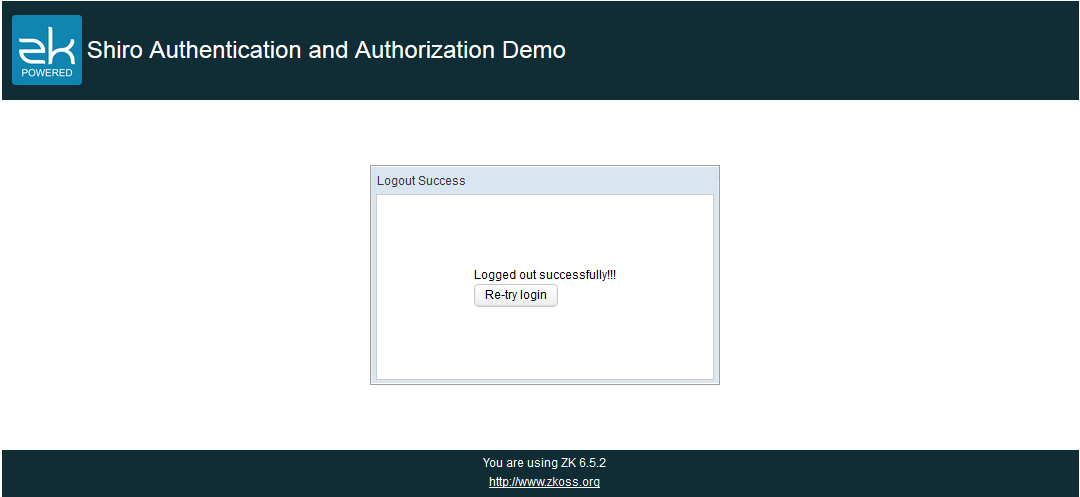
**Figure 15:**  Marketing Admin Console page.



**Figure 16:**  Sales Admin Console page.



**Figure 17:**  Products Admin Console page.



**Figure 18:**  Logout page. User gets here after clicking the logout link (see Figures 15, 16, 17) in upper right corner after logged in.

**Code Review:**

* **Authentication**

This is implemented by the login page in Figure 12 using the Apache Shiro framework. For a quick introduction of how to secure ZK applications with Apache Shiro, see this tutorial [here](http://books.zkoss.org/wiki/Small_Talks/2012/March/Securing_ZK_Applications_With_Apache_Shiro). The tutorial basically shows how to quickly implement a login page with ZK using Apache Shiro and protects a set of urls so that only logged in user having the right roles can access those urls. That is, each protected url requires a set of roles before it can be accessed.

If you try to use the latest Apache Shiro version, then the techniques from tutorial won’t work out of the box. Also, the tutorial doesn’t show how to do authentication against a SQL database. So I will show how to do both of these … stay tuned.

What configuration is required to make Shiro work with ZK and SQL? Two places, web.xml and a properties file named shiro.ini placed in classpath. First of all, web.xml file. The ZK tutorial used the following in web.xml for Shiro 1.1.0.

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.IniShiroFilter</filter-class>

</filter>

…

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

If you want to use the latest Shiro at version 1.2.2 as of this writing, then sorry, it’s not going to work. What I did find working is

<listener>

<listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>

</listener>

…

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

<dispatcher>INCLUDE</dispatcher>

<dispatcher>ERROR</dispatcher>

</filter-mapping>

For details, see the web.xml in source code. Now shiro.ini file. This is placed in WEB-INF folder since it’s in the classpath. For a more detailed guide on using shiro.ini, see [Apache Shiro Configuration guide](http://shiro.apache.org/configuration.html).

Note the shiro.ini from ZK Shiro tutorial hard codes the username, password and roles associated with each username in [users] section, i.e.

[users]

# <username> = <password>, <role >

admin = a,administrator

marketingguy = a,marketing

productsguy = a,products

salesguy = a,sales

Now since we are aiming for production quality by using SQL, we used these instead

# connection pool settings

jdbcRealm = org.apache.shiro.realm.jdbc.JdbcRealm

jdbcRealm.dataSource = $ds

**jdbcRealm.userRolesQuery** = select sec\_role.role\_name from security\_role sec\_role, security\_user sec\_user, security\_role\_user sec\_role\_user where sec\_user.user\_login\_name = **?** AND sec\_role\_user.user\_id = sec\_user.user\_id AND sec\_role\_user.role\_id = sec\_role.role\_id

**jdbcRealm.authenticationQuery** = select user\_password from security\_user where user\_login\_name = **?**

securityManager.realms = $jdbcRealm

# JDBC settings

ds = com.mchange.v2.c3p0.ComboPooledDataSource

ds.driverClass = com.mysql.jdbc.Driver

ds.jdbcUrl = jdbc:mysql://localhost:3306/zkfuse

ds.user = root

ds.password = password

**Listing 3**

Note the lines starting with *jdbcRealm.userRolesQuery* and *jdbcRealm.authenticationQuery.* That’s where the magic takes place.

*jdbcRealm.userRolesQuery* will define the SQL query to retrieve a list of user’s roles by using user’s login name as input parameter of SQL query for authorization. So after a user logs in successfully, if the user tries to visit an url without the required role, Shiro will redirect user to the access denied page.

So where are these url with required roles and access denied page specified in shiro.ini?

Access denied page is defined in [main] section of shiro.ini from my demo along with other things as

[main]

sampleauthc = shiro.sample.SampleFormAuthenticationFilter

sampleauthc.loginUrl = /security/shiro\_login.zul

sampleauthc.usernameParam = user

sampleauthc.passwordParam = pass

sampleauthc.successUrl = /security/shiro\_index.zul

sampleauthc.failureKeyAttribute=loginFailure

**roles.unauthorizedUrl = /security/accessdenied.zul # Access denied page**

Note the Java class SampleFormAuthenticationFilter, it’s used to override FormAuthenticationFilter class from Shiro to set error message as the value of an attribute named loginFailure (specified in sampleauthc.failureKeyAttribute entry above) in javax.servlet.ServletRequest after failed login. So when the login page is displayed and if this attribute is present, a custom error message defined in login.zul will be shown to the user

A list of url with the required role(s) is specified in the [urls] section of shiro.ini from ZK tutorial as

[urls]

/login.zul = sampleauthc

/marketing/\*\*=sampleauthc, roles[marketing] # ‘marketing’ role required to access this url pattern

/products/\*\* = sampleauthc, roles[products] # ‘products’ role required to access this url pattern

/sales/\*\* = sampleauthc, roles[sales] # ‘sales’ role required to access this url pattern

/zkau/\*\* = anon # ‘anonymous’ role. Anyone can access this url pattern

/home.zul = anon # ‘anonymous’ role. Anyone can access this url

Whoa, not so fast. The [urls] section of shiro.ini from my demo only has

[urls]

/security/shiro\_login.zul = sampleauthc

That’s because I am using the ajax feature of ZK to display those pages that should be restricted by roles of user. So when the user clicks on the left navigation bar to show those restricted pages, there is no URL change at all! So the real authorization of matching user’s roles against the roles required to access a restricted page will be done in Java.

In effect, *jdbcRealm.userRolesQuery* line does nothing right now in my demo. Then what’s the point of showing it here? Well, I didn’t know it wasn’t going to work back then when I was implementing it and I didn’t bother or forgot to delete that line from shiro.ini. But still it serves as a good reference when you want Shiro to restrict access by url.

The real meat of how authorization is done will be discussed in next section. Right now, let’s get back to authentication (which is what this section is about).

*jdbcRealm.authenticationQuery* entry in shiro.ini specifies the SQL for authentication (for real this time) by using the username as input parameter of SQL query. After successful login, user will be redirected to the index page specified by this line in [main]

section of shiro.ini: sampleauthc.successUrl = /security/shiro\_index.zul

If not, then displays error message in login form. And that’s it. No need for any other config or Java code apart from implementing the frontend zul files.

* **Authorization**

This section will discuss how authorization is implemented using Java. If you wander why Shiro isn’t used to do this, see Authentication section above.

The class that does real authorization is ShiroAuthorizationUtil. This class has a public static isAuthorized(..) method that takes both pageId (ID of page user is accessing) and username (login name) as input parameters and returns true if the user has the required roles to access the target page.

So where is this ShiroAuthorizationUtil.isAuthorized(..) called and where is this page ID defined? The answer to both is in the controller. Now this controller is THE controller from MVC design pattern. Basically, an abstract controller class ShiroBaseController is created to extend ZK’s standard controller class SelectorComposer such that whenever a subclass of ShiroBaseController (which corresponds to a restricted page) is accessed, the subclass will:

1. Check to see if user is authenticated, if not, redirect to access denied page. If authenticated, go to 2.
2. Check to see if user is authorized to access the page by calling ShiroAuthorizationUtil.isAuthorized(..).

If authorized, proceed to render the page. If not, redirect to access denied page

Each subclass of ShiroBaseController will also define the page ID as private static final String so isAuthorized(..) should never have empty page ID.

Remember we implemented the DAO and Service layers for persisting data in Figure 8 in Security Admin Console demo? That effort is going to pay off now. In isAuthorized(..) method, we use the username name to retrieve the SecurityUser object (which maps to security\_user table). This object contains method getSecurityRoles() that can be used to return a list of SecurityRole objects (which map to security\_role table).

This list of SecurityRole objects is then converted to a list of strings representing user roles. The user roles are then compared against the roles required to access the page.

Note currently, the roles required to access a page is implemented as ‘any of’, meaning all it takes to access the page is for the user to have any one of the roles required.

So where is the set of roles defined for a restricted page? In the properties file *pagId-roles.properties* as

HOME\_PAGE=ADMIN\_ROLE,MARKETING\_ROLE,SALES\_ROLE,PRODUCTS\_ROLE

MARKETING\_PAGE=ADMIN\_ROLE,MARKETING\_ROLE # ForMarketing Admin Console page

SALES\_PAGE=ADMIN\_ROLE,SALES\_ROLE # For Sales Admin Console page

PRODUCTS\_PAGE=ADMIN\_ROLE,PRODUCTS\_ROLE # For Product Admin Console page

From above, we can see page ID *PRODUCTS\_PAGE* requires user to have any of the ADMIN\_ROLE or PRODUCT\_ROLE roles to access the Product Admin Console page. For details, see ShiroAuthorizationUtil for implementation.

Now if you examine the doBeforeCompose(..) method of ShiroBaseController, you will notice that the code to verify a user is authenticated is a simple two liners using Shiro’s API

Subject subject = SecurityUtils.getSubject();

// Redirect to access deny page if not authenticated

if ( !subject.isAuthenticated() ) {

…

}

Now imagine trying to do the same thing using JAAS or Spring Security API.

Finally, for fine-grained authorization, we would also want to use Permissions attached to the Role. These info can be retrieved quite easily from the getSecurityPermissions() method of SecurityRole object. The permissions can then be used to decide what operations a user can do in the restricted page. But for simplicity, it’s not used in demo.

**More Shiro stuff**

This section will provide more info about Shiro so reader can get a feel of what’s the best practice for Shiro is and whether it’s ready for production.

* [Apache Shiro Community Forums](http://shiro.apache.org/forums.html)
* [Apache Shiro combined with LDAP](http://stackoverflow.com/questions/3422012/apache-shiro-combined-with-ldap)
* [Apache Shiro, is it ready for Java EE 6? (a JSF2-Shiro Tutorial)](http://balusc.blogspot.co.nz/2013/01/apache-shiro-is-it-ready-for-java-ee-6.html)
* [Java Web Application Security - Part III: Apache Shiro Login Demo](http://raibledesigns.com/rd/entry/java_web_application_security_part2)

**Best practice for roles and permissions**

* [Changing permissions at runtime](http://shiro-user.582556.n2.nabble.com/Changing-permissions-at-runtime-td3282662.html)
* [Retrieving The List Of Permissions (Or Roles) For A User](http://shiro-user.582556.n2.nabble.com/Retrieving-The-List-Of-Permissions-Or-Roles-For-A-User-td6634613.html)
* Ask the Shiro forums!

***Demo 4 – JVM Information***

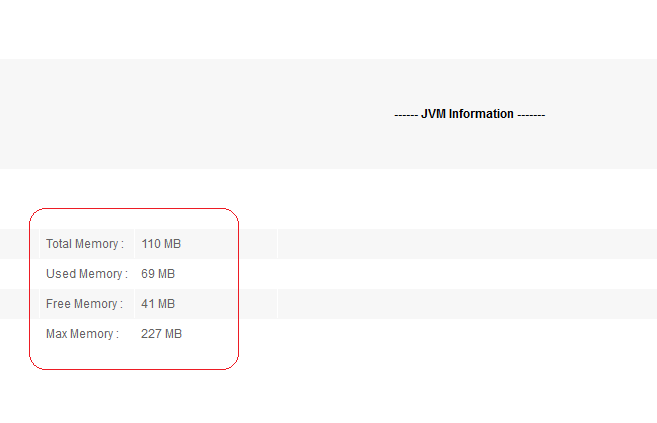
**Objective:**

Implement a simple page that displays the various memory settings (i.e. total, free, used and maximum) of JVM running the web application.

**Motivation:**

Oftentimes, the web application may appear to be running slow. Then you are left wandering if JVM is short of memory (memory leak?) or if something else is going on. Being able to see the memory settings of JVM can then prove to be useful for troubleshooting.

**Frontend Designs:**



**Figure 18:**  Memory settings of JVM.

**Solution Design:**

The controller used to calculate JVM memory settings is JVMInfoController. Frontend zul file is jvmInfo.zul.

The code used to calculate memory settings of JVM is actually quite simple.

int mb = 1024 \* 1024;

Runtime runtime = Runtime.getRuntime();

Long usedMemoryVal = (runtime.totalMemory() - runtime.freeMemory()) / mb; // Used Memory

Long freeMemoryVal = runtime.freeMemory() / mb; // Free Memory

Long totalMemoryVal = runtime.totalMemory() / mb; // Total Memory

Long maxMemoryVal = runtime.maxMemory() / mb; // Maximum Memory

You may be wandering the difference between maxMemory() and totalMemory(). Well, this [link](http://stackoverflow.com/questions/3571203/what-is-the-exact-meaning-of-runtime-getruntime-totalmemory-and-freememory) should answer it all.

Basically when you run a Java process named Foo like ‘java -Xms64m -Xmx1024m Foo’, then maxMemory() will be 1024MB.

***Demo 5 - Web Service Security Demo***

**Objective:**

Showcase the use of WS-Policy (or [**WS-SecurityPolicy**](http://en.wikipedia.org/wiki/WS-SecurityPolicy)) within WSDL to implement Web Service Security by using the [**CXF framework**](http://cxf.apache.org/)

**Solution Design:**

This demo will allow user to select a web service from the ‘Web Service’ drop down list, then the Reference label, Description and WSDL text areas will be updated via ajax to display the reference (if any), description and WSDL of the selected web service respectively. See Figure 19 below.

After entering a number in the input field then click the Send button, the Soap Request and Soap Response text areas will then be updated via ajax to show the soap request and response created in real time. See Figure 19 below.

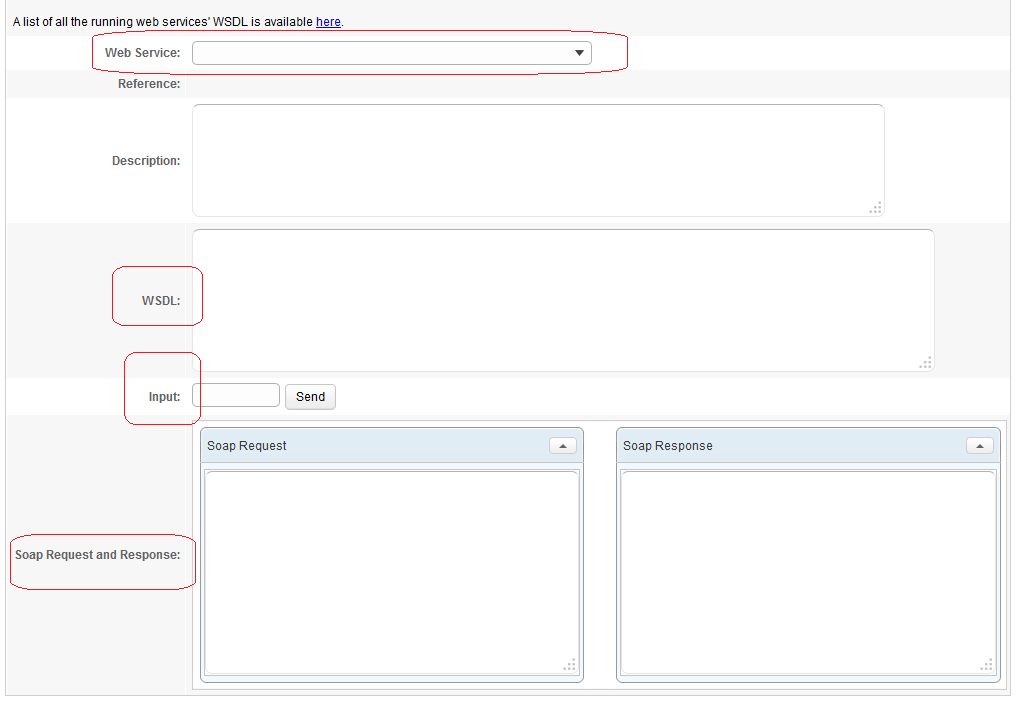
All the web services from the drop down list are the same and running in this demo app except they all have different WS-Policy (or WS-SecurityPolicy) specified in the WSDL for different usage scenarios and different xml configuration for encryption and signing. The WS-Policy basically acts a reusable template for Web Service security.

What do all these web services do? Well, they simply take the number from Input field (text box) then return the doubled number in soap response.

Credits for these web services go to Glen Mazza and it's available for download [here](https://github.com/gmazza/blog-samples). One can read more about how to set up and run these as standalone web services in his blog [here](http://www.jroller.com/gmazza/entry/cxf_x509_profile).

Since this is a WSDL first web service, the WS-Policy used can be seen in browser from WSDL’s url (i.e. http://...?wsdl). The WSDL is also visible in the WSDL text area once a Web Service is selected from the Web Service drop down list.

Basically, different WS-Policy can be applied to WSDL of the same web service to generate different soap request with minimal code and configuration changes, depending on usage scenarios. See Figure 19 for details.



**Figure 19:** Web Service Security Demo

* WS-Policy (or WS-SecurityPolicy)

One thing all these web services have in common is they all use [UsernameToken](http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf) profile (except for X509 asymmetric message protection). This is the de factor web service profile for username/password authentication

So where exactly is WS-SecurityPolicy specified in WSDL? Well, they are usually specified at bottom of WSDL in sections highlighted bold below:

<?xml version='1.0' encoding='UTF-8'?>

<wsdl:definitions ...>

...

**<wsp:Policy wsu:Id="DoubleItUTSymmetricX509Policy">**

**...**

**</wsp:Policy>**

**<wsp:Policy wsu:Id="DoubleItBinding\_DoubleIt\_Input\_Policy">**

**...**

**</wsp:Policy>**

**<wsp:Policy wsu:Id="DoubleItBinding\_DoubleIt\_Output\_Policy">**

**....**

**</wsp:Policy>**

</wsdl:definitions>

As you can see, we can have multiple policies but then how does the soap request or response know which one to use? By using the wsp:PolicyReference element (with namespace http://www.w3.org/ns/ws-policy) to refer to wsu:Id element (with namespace <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd>) specified in WS-Policy above as below within WSDL:

<wsdl:binding name="DoubleItBinding" type="tns:DoubleItPortType">

**<wsp:PolicyReference URI="#DoubleItUTSymmetricX509Policy"/>**

<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="DoubleIt">

<soap:operation soapAction="http://www.example.org/contract/usernametoken\_x509\_symmetric/DoubleIt/DoubleItPortType/DoubleItRequest"/>

<!-- soap request -->

<wsdl:input>

<soap:body use="literal"/>

**<wsp:PolicyReference URI="#DoubleItBinding\_DoubleIt\_Input\_Policy"/>**

</wsdl:input>

<!-- soap response -->

<wsdl:output>

<soap:body use="literal"/>

**<wsp:PolicyReference URI="#DoubleItBinding\_DoubleIt\_Output\_Policy"/>**

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

Note how the elements *wsp:PolicyReference* highlighted in bold above are referring to *wsu:Id* elements.

Since web service security is a big topic, I won’t go into details of what WS-Policy looks like or how WS-Policy works for each of the Web Service Security scenarios listed below. Instead I will give a brief description what each scenario is used for and provide more links along the way and in the *References* section.

* Web Service Security scenarios available for selection from the *Web Service* drop down list (Figure 19) are:

1. [**UsernameToken with hashed password**](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.express.doc%2Fae%2Fcwlp_wssec_templates_scenario1.html)

This WS-SecurityPolicy template will transmit message in plain text but protects password by hashing or digesting the password with SHA algorithm. In production, this template should be used with SSL/TLS to protect message at transport layer.

Note the IBM link provided above shows the use of WS-Policy to achieve UsesrnameToken with hashed password **AND timestamp over SSL** while my demo doesn’t use SSL in WS-Policy at all. This is because I want to display soap request with hashed password in clear text in the *Soap Request* text area (Figure 19). With SSL, we wouldn’t be able to see the soap request and response as everything will be encrypted.

An example of what the usernametoken with hashed password in soap request looks like

<soap:Header>

<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soap:mustUnderstand="1">

<wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-49">

<wsse:Username>joe</wsse:Username>

<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest">**c0H4fASfgQWrAnuaezVOyQXRj6k=**</wsse:Password>

<wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">ClyV00hzoxElwkPZSSeQvA==</wsse:Nonce>

<wsu:Created>2013-09-29T21:38:14.716Z</wsu:Created>

</wsse:UsernameToken>

</wsse:Security>

</soap:Header>

The highlighted text in bold is the hashed password. Of all the web services, this is the only one where you get to see the actual usernameToken element within soap header and see the doubled integer in the body of soap response. This is because in all other web services, the usernameToken element in soap request and the doubled integer in soap response are all encrypted as data of **CipherValue** element within **EncryptedData** element, e.g.

**<xenc:EncryptedData** **xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="ED-72" Type="http://www.w3.org/2001/04/xmlenc#Element">**

<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>

<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

<wsse:SecurityTokenReference xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey">

<wsse:Reference URI="#EK-BE8558EEE285F7A1CB138053125172948"/>

</wsse:SecurityTokenReference>

</ds:KeyInfo>

<xenc:CipherData>

**<xenc:CipherValue>**4hW3NtYReBfE0puQnE3JOtFDC1wL+8FwziuWgVzXYWnquEbAd+GptGfrm6hccpMV2xyI6L+xpL2XrcL6RGVu82Bp5jKIP0jSoAMYcvqFnMpKUcWDAFstsvMq3tPa4rDYtbVhwSEH0VaMp7sNNZOyKLEO5Dv4cjg1xH9mz+fEDE13bg3y/zSBZ0CvwTyARvtJpkwhHQYsC2pWS3RySKwp3Hxn977MV4rm88cu/U/2ZCwqZ7sxz1t7FLg6iXZFaazBtcSC3AYK+qt+DhJy7L2gmzAeMcfXzfoC30BT+MZnyKusRqncSTuUpmGVEDdSbbaG88KxdtiLms7zIPP2nYjhnXVkUnSuqIP/2k90e7ueP5Q=**</xenc:CipherValue>**

</xenc:CipherData>

**</xenc:EncryptedData>**

The exact soap request and response body parts to sign and encrypt are specified in WS-Policies.

1. **X509 asymmetric message protection**

This policy template is best used if the client must authenticate itself to the service with only X509 client certificate and if the message exchange must be signed and encrypted.

Note this binding is exactly the same as “*UsernameToken with X509 asymmetric message protection*” except that UsernameToken won't be used for authentication in this case.

1. [**UsernameToken with X509 symmetric message protection**](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.express.doc%2Fae%2Fcwlp_wssec_templates_scenario6.html)

This WS-SecurityPolicy template is best used if the client can use only a UsernameToken to authenticate itself, and if the message exchange must be signed and encrypted without using client's certificate.

Basically, the client generates the Symmetric Key which is used to sign and encrypt the request. The symmetric key is then encrypted with the public key of the server's certificate before sending the request along with encrypted symmetric key to the server. The server decrypts the received symmetric key with its own private key (only the private key can decrypt any message encrypted with its paired public key), and uses the symmetric key to decrypt and verify the signature of the request.

It should become clear the advantage of using symmetric binding over asymmetric binding is that the server doesn't need to import the public key of X509 certificate of all the clients using its web service into its keystore (which is required for X509 asymmetric binding). Imagine the web service is going to be used by 100 clients then the server will have to import and manage 100 public certificates from clients into its keystore!!

Note WS-SecurityPolicy can be used to specify which parts of request/response message to encrypt and sign in WSDL. By default, the message is signed first before encrypted but this can be overwritten to encrypt first before signing using WS-SecurityPolicy.

1. [**UsernameToken with X509 asymmetric message protection**](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.express.doc%2Fae%2Fcwlp_wssec_templates_scenario4.html)

This WS-SecurityPolicy template is best used if the client must authenticate itself to the service with both an X509 client certificate and a UsernameToken.

An X509 certificate has a pair of keys, public and private. The public key will be sent to any party that wants to communicate with you using *X509 asymmetric binding* BUT NEVER the private key (else game over). Now for any message encrypted with the public key, only the private key can be used to decrypt it. And for any message signed with the private key, the same public key can be used to verify that the xml signature has indeed been signed by its paired private key. Confused? See below.

Basically the client will sign the message with its own private key then encrypts the message with server's public key before sending the request to the server. The server will decrypt the message with its own private key (since message encrypted with server’s public key) then verifies the xml signature of client using the client's public key.

Response from server to client will use the same mechanism except the message will be signed with server's private key and encrypted with client's public key. The client will decrypt the message with its own private key then verifies the xml signature of server using the server's public key.

Note WS-SecurityPolicy can be used to specify which parts of request/response message to encrypt and sign in WSDL. By default, the message is signed first before encrypted but this can be overwritten to encrypt first before signing using WS-SecurityPolicy.

Still confused? Then this [Basic Signing and Encryption Scenario link](https://access.redhat.com/site/documentation/en-US/JBoss_Fuse/6.0/html/Web_Services_Security_Guide/files/MsgProtect-SOAP-SignEncryptScenario.html) with nice diagram will definitively clear things up. Note the order of encryption and signing definitively matters. The default order is to sign before encrypting. But if you include this element ***sp:EncryptBeforeSigning*** in your symmetric policy, the order is changed to encrypt before signing. See this [link](http://fusesource.com/docs/esb/4.4/cxf_security/MsgProtect-SOAP-SymmetricPolicy.html) for reference.

**References:**

# [WS-SecurityPolicy 1.2 Spec](http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/errata01/os/ws-securitypolicy-1.2-errata01-os-complete.html) or see [this](http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html) to look for latest spec from OASIS

* A list of [WS-SecurityPolicy and templates](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.express.doc%2Fae%2Fcwlp_wssec_templates_scenario4.html) from IBM
* [Protection of web services with an X.509 token](http://pic.dhe.ibm.com/infocenter/radhelp/v9/index.jsp?topic=%2Fcom.ibm.websphere.wlp.nd.multiplatform.doc%2Fae%2Fcwlp_wssec_x509.html) from IBM
* [Understanding WS–Security Policy Language](http://wso2.com/library/3132/) from WSO2
* [Java web services: Modeling and verifying WS-SecurityPolicy](http://www.ibm.com/developerworks/webservices/library/j-jws21/index.html) from IBM

**Code Review:**

Now comes the tricky parts of documenting how CXF is used to implement all these. I can see why developers like to document things in their heads.

When it comes to implementing web services using the CXF framework, we can have [Code First](http://cxf.apache.org/docs/developing-a-service.html#DevelopingaService-JavaFirstDevelopment) or [Contract First](http://cxf.apache.org/docs/developing-a-service.html#DevelopingaService-WSDLFirstDevelopment) (i.e. WSDL First). Code first approach is handy if we want to generate WSDL from Java.

The implementation of WS-Security (i.e. web service security) can come down to using the [CXF Interceptor](http://fusesource.com/docs/esb/4.2/cxf_interceptors/CXFInterceptorIntro.html) approach (*Configuring the WSS4J Interceptors* section of this WS-Security [link](http://cxf.apache.org/docs/ws-security.html) from CXF) with WSS4J (like code first) or simply use the WS-SecurityPolicy support of CXF by using the WS-Policy to define security policies to use within WSDL (like contract first). From what I read, [the best security practice](http://ws.apache.org/wss4j/best_practice.html) is to use the WS-SecurityPolicy as it’s easier to use and gives you more automatic protection against various attacks while the CXF Interceptor approach will require you to take care of those yourself by manual coding.

This demo is using the WSDL first approach to generate both the server and client code from WSDL for each web service. For WS-Security, the WS-SecurityPolicy approach is used. Spring is then used to configure the various security settings (e.g. keystore and WSDL locations, name of encryption key, …) for both server and client for each web service. Easy ehh!

* Specify WS-SecurityPolicy in WSDL

If you have read the links from Solution Design section above, then you should have some idea of how to create the WS-Policies of web services within WSDL, in case you are not confused enough, I am going to throw more links at you:

* [Specifying an Authentication Policy](http://fusesource.com/docs/framework/2.4/security/Auth-Policy.html)

### [Specifying the Algorithm Suite](http://fusesource.com/docs/framework/2.4/security/MsgProtect-SOAP-SpecifyAlgorithmSuite.html): may need to refer back to this if want to confirm the algorithm used when testing

### [Specifying a Symmetric Binding Policy](http://fusesource.com/docs/esb/4.4/cxf_security/MsgProtect-SOAP-SymmetricPolicy.html)

### [Specifying an AsymmetricBinding Policy](http://fusesource.com/docs/esb/4.2/service_security/MsgProtect-SOAP-AsymmetricPolicy.html)

* Check the WS-Policies used in Glen’s tutorials [here](https://github.com/gmazza/blog-samples)
* Sometimes, you may run into a problem where you expect the WS-Policy to work but it doesn’t and you can’t figure out why. If searching forums and stackoverflow turns up nothing, then post questions to CXF forum! The CXF forum is very active and the committers always get back to you within days!

While I was implementing this demo, many tricky problems got solved through forums. I would even go as far to say subscribing to these forums then use a decent email client like Mozilla Thunderbird to organize emails from these forums into its own folders is a must. Some of the security stuff is way too technical to figure out on your own.

To test what you just changed in WS-Policy is working,

* Whenever the WS-Policy or WSDL is changed within WSDL, do a build to run code generation below
* Change the corresponding Callback Handler and Spring config files as required

Whenever the port number of deploy environment is changed, the port number within WSDL also needs to change as well then do a new build for the change to take effect, E.g.

<wsdl:service name="DoubleItService">

<wsdl:port name="DoubleItPort" binding="tns:DoubleItBinding">

<soap:address location="http://localhost:**8080**/zkfuse\_web/wsservice/usernametoken\_hashed\_password/doubleit" />

</wsdl:port>

</wsdl:service>

Note the localhost name is fine for development but may need to change to the hostname or ip address of the target server (hence new build) to work.

* Code Generation

How is the code generated from WSDL for both server and client? Well, in zkfuse\_web\_service module, the WSDLs are in

* \src\main\resources\un\_hashed\_password\DoubleIt.wsdl defines the WSDL for *UsernameToken with hashed password* web service
* \src\main\resources\un\_x509\_asym\_webservice\DoubleIt.wsdl defines the WSDL for *UsernameToken with X509 asymmetric message protection* web service
* \src\main\resources\un\_x509\_sym\_webservice\DoubleIt.wsdl defines the WSDL for *UsernameToken with X509 symmetric message protection* web service
* \src\main\resources\x509\_asym\_webservice\DoubleIt.wsdl defines the WSDL for *X509 asymmetric message protection* web service

Then the Maven plugin [cxf-codegen-plugin](http://cxf.apache.org/docs/maven-cxf-codegen-plugin-wsdl-to-java.html) from CXF is used to generate Java artefacts of all web services as part of the Maven build process (i.e. mvn install). Search cxf-codegen-plugin in \zkfuse\pom.xml and \zkfuse\zkfuse\_web\_service\pom.xml for how it’s configured.

* Callback Handlers

Both server and client code need to implement the javax.security.auth.callback.CallbackHandler to provide the password used if UsernameToken profile is used. For client code, the Spring client side configuration file is actually capable of providing both username and password used for UsernameToken profile (i.e. see ws-security.username and ws-security.password from [here](http://cxf.apache.org/docs/ws-securitypolicy.html)) but hard coding client’s password to access a web service may not be a good idea for production.

The server callback handler also needs to provide the password for the private key identified by the alias used in decryption/signing. Again, these passwords can actually be specified in Spring configuration file. We will come to these later.

For production, the server callback handlers should use the password from User account (e.g. SecurityUser object from Security Admin Console) to set the password property of WSPasswordCallback object with matching username, e.g. see ServerPasswordCallback from the package org.zkfuse.webservice.service.ut.hashedpassword for server callback handler of the UsernameToken with hashed password web service.

* Server implementation

The service provided on server side is implemented by the DoubleItPortTypeImpl class, which implements a DoubleITPortType interface generated from the WSDL. The only business logic we provide right now is to return the doubled integer. The integer to double is from input text box.

* Spring Configurations

The Spring configurations for both server and client web services are in \zkfuse\_web\_service\src\main\resources\spring\. You can see for each web service, there is one spring config file for client and one spring config file for server. For details of how to configure, see [WS-SecurityPolicy](http://cxf.apache.org/docs/ws-securitypolicy.html) from CXF and [Providing Encryption Keys and Signing Keys](http://fusesource.com/docs/framework/2.4/security/MsgProtect-SOAP-ProvideKeys.html) from FuseSource.

Now which spring context file imports all the spring configurations defined above? It’s applicationContext-ws.xml.

That should cover most of the things for this demo. Of all of the demos, troubleshooting web services has to be the most difficult and tricky ones. So good luck. ☺

***Demo 6 – Encryption Demo***

**Objective:**

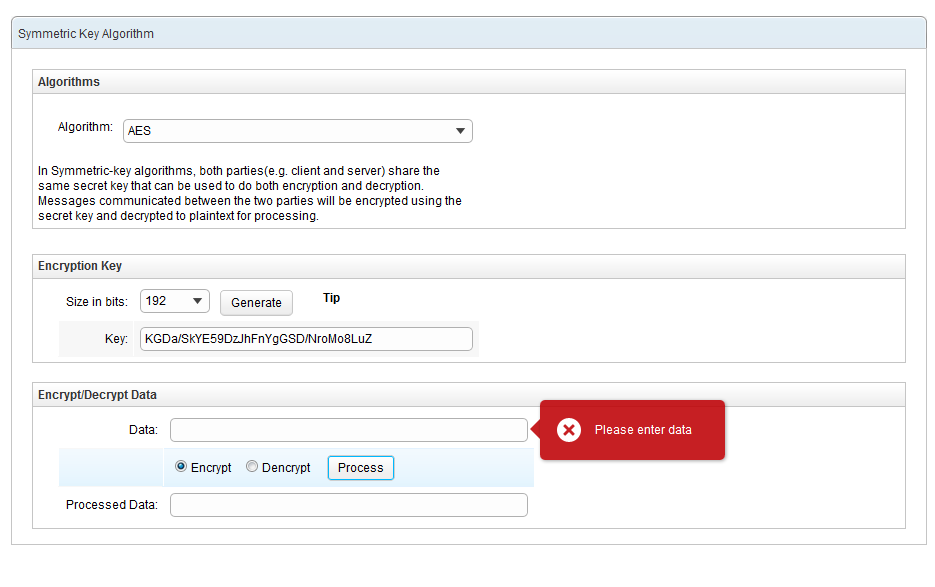
Show how easy it is to implement encryption and decryption of three algorithms: **AES**, **Twofish** and **Rijndael**.

**Motivation:**

Of the three algorithms above, [AES](http://en.wikipedia.org/wiki/Advanced_Encryption_Standard) (a variant of Rijndae) and [Twofish](http://en.wikipedia.org/wiki/Twofish) (successor of Blowfish) are widely used today in real world for commercial products. The marketing team of software companies may claim their software is secure as it uses military grade encryption AES 128/192/256 to do such and such. And yes, I did see such marketing.

Note AES is approved, endorsed and adopted by NSA. As a result, it’s an industry standard. So whether it’s truly secure in terms of whether they are capable of breaking it, well, your guess is as good as mine. ;)

**Frontend Designs:**



**Figure 20:** Encryption/Decryption panel

**Solution Design:**

To encrypt/decrypt data, three steps are required:

1. Select an algorithm from the Algorithm drop down list
2. Generate the encryption key (i.e. binary data encoded as ACII strings by [Base64](http://en.wikipedia.org/wiki/Base64) encoding) by selecting an encryption key size in bits from the ‘Size in bits’ drop down list then click the Generate button. Note only one of three key sizes is allowed: 128, 192 and 256 bits.
3. To encrypt/decrypt data, enter data in Data text field. Select Encrypt/Decrypt radio button then click the Process button to encrypt/decrypt the data entered in Data text field. See Figure 20 above.

Note if any of the required fields is missing when an UI event takes place (e.g. click Process button), then an error message should pop up beside the mandatory field as in Figure 20.

**Code Review:**

The controller used here is EncryptionDemoController. Frontend zul file is encryption\_demo.zul.

The open source library [Bouncy Castle](http://www.bouncycastle.org/) is used to implement the encryption and decryption of all three algorithms here. The class that does all these is EncryptorUtil in zkfuse\_util module.

Note it may be necessary to install JCE (Java Cryptography Extension) from [here](http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html) before you can get encryption working. Install it anyway. You will need it to do encryption or SSL for key size beyond a certain length (e.g. 128 bits?) latter. Instruction to install it into JDK is in the download.

If you are wandering the spec (e.g. allowed key length or parameter) for an algorithm or what other algorithms are available in Bouncy Castle, then check [here](http://www.bouncycastle.org/specifications.html).

Now when it comes to coding cryptography using Java, one thing I have to emphasize is ‘*Don’t copy and paste code from the web to make things work without fully understanding what’s going on*’! That’s the formula for creating an insecure system (the problem is somewhere between the chair and computer).

Or if you really have to do that, make sure you use a popular, battle tested API with clear sample code showing how to use it, e.g. Bouncy Castle. This [thread](https://forums.oracle.com/thread/1527955) shows the danger of copy and paste code from web.

One concept that may not be obvious in EncryptorUtil class is that all encryption key, encrypted and decrypted data are binary data in bytes as in 0100101011100… So to store and represent these encrypted data and encryption key, Base64 encoding is used to convert bytes into ACII characters. For decrypted data, we can simply convert it back to original string as ‘new String( rv ).trim()’ where rv is the byte array (i.e. byte[]) of decrypted data.

The cool, red error notification box in Figure 20 is done by the line

|  |
| --- |
|  |

Clients.showNotification(msg, type, component, position, duration);

See this [link](http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/UI_Patterns/Useful_Java_Utilities) for how to configure this notification box in various positions.

That’s the end of walkthrough for all tutorials. Hope you find it interesting. Feel free to give feedback. ;)