Day5: ZK Application

Michael Mountrakis
mountrakis@illumineit.com

A training seminar given for MOU S.A.

Athens, December 2015

Day 5

1 ZK Application ←
2 ZK Example Application – Extended Lab
3 Enterprise Java Project Implementation Directives

ZK Application

Overview

- A simple example
- Specifications
- Designing the interface
- Implementing Model
- Implementing the Controller
- Applying basic security
- Methodology template

Authoring – reviewing system

- An authoring – reviewing system
  - Authors, Reviewers, Documents
  - Authors write Documents
  - Reviewers review the documents and write some comments

Functional specifications

- Author
  - Name, surname, email
  - Can be inserted, updated, deleted, searched.
  - Must be system persisted.
- Reviewer
  - Name, surname, email
  - Can be inserted, updated, deleted, searched.
  - Must be system persisted.

Functional specifications

- Document
  - XML text based text, title, created, author
  - Can be inserted, updated, deleted, searched.
  - Must be system persisted.
- A Review
  - One document, one reviewer, reviewer comment, review date
  - Can be inserted and searched only.
  - Must be system persisted.

Functional specifications

- Access - Authority
  - Only registered users.
  - Author cannot be a Reviewer
  - Reviewer cannot be an Author
- Document insertion
  - A document can be also supplied by a third party system only by a known author. This results to a new document only.

Technical specifications

- Implementation technology Oracle JVM 7.0
- Implementation of UI: ZK 5
- Inserted Document commits to valid XML 1.0 text
- Information Persistance on MariaDB Rdbms server of latest version
- Application Server WebLogic 11g
- Third party connectivity SOAP 1.2

Design

- How I start?
  - Review the tools from tecnhical specs:
    - Oracle java 1.7, Eclipse IDE, ZK Framework 5.0, WebLogic 11g server.
  - Design approach:
    - DB first: E-R first
    - Middleware: JDBC / DAO – Entity pattern
    - Web tier
    - W-S tier

DB - Model

- Major Decision:
  - JPA or JDBC ?
    - JPA if project designed from the start and has to be DB agnostic
    - JDBC if DB is fixed.
- Design your database
- Implement your database

DB – Model in Java. When JDBC

- Use Illumine´s JetEngine 2.0 tool to generate DAO/Entity classes from your DB:
  - Freeware – no fees no licenses.
  - Opensource
  - Simple code that you can modify
  - Supports ORACLE, MySQL, MariaDB, PostGre
  - DAO/Entity pattern
  - Non depended only JDBC drivers included
  - On site support.

DB – Model in Java. When JPA

- Hibernate
  - Matured
  - Open standard
  - Well documented

Java Application Design

Steps

- Derive mapping of application logic to screens
- Derive navigation scheme
- Design each screen
- Implement basic controllers for each screen
- Group similar screen functionality – modify application to re-use components
- Apply security

Application logic to screens

Decoupled mapping:

- Each application entity as this is implemented on BackEnd should have a Web management screen.
- Each Web screen should be implemented from a single ZUL file. Example:
  - Backend: Author --> Web: author.zul
  - Backend: Document --> Web: document.zul
  - Backend: Reviewer --> Web: reviewer.zul
  - Backend: Review --> Web: review.zul

Application logic to screens

Decoupled mapping:

- Each application entity as this is implemented on BackEnd should have a Web management screen.
- Each Web screen should be implemented from a single ZUL file. Example:
  - Backend: Author --> Web: author.zul
  - Backend: Document --> Web: document.zul
  - Backend: Reviewer --> Web: reviewer.zul
  - Backend: Review --> Web: review.zul

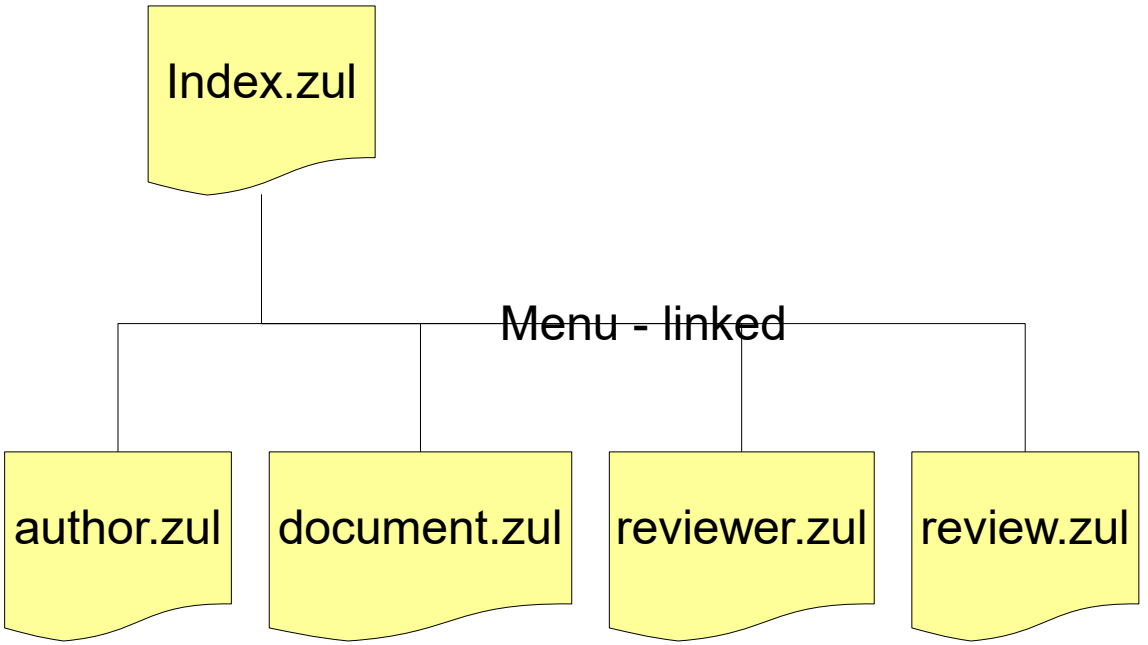
Application logic to screens

Advice:

- Dont overload functionality: *simple screens*
- Always have a clear vision: *simplicity – Isolate functional blocks.*
- Design only from the *functional perspective* – the *Web UI designer* will do the rest.

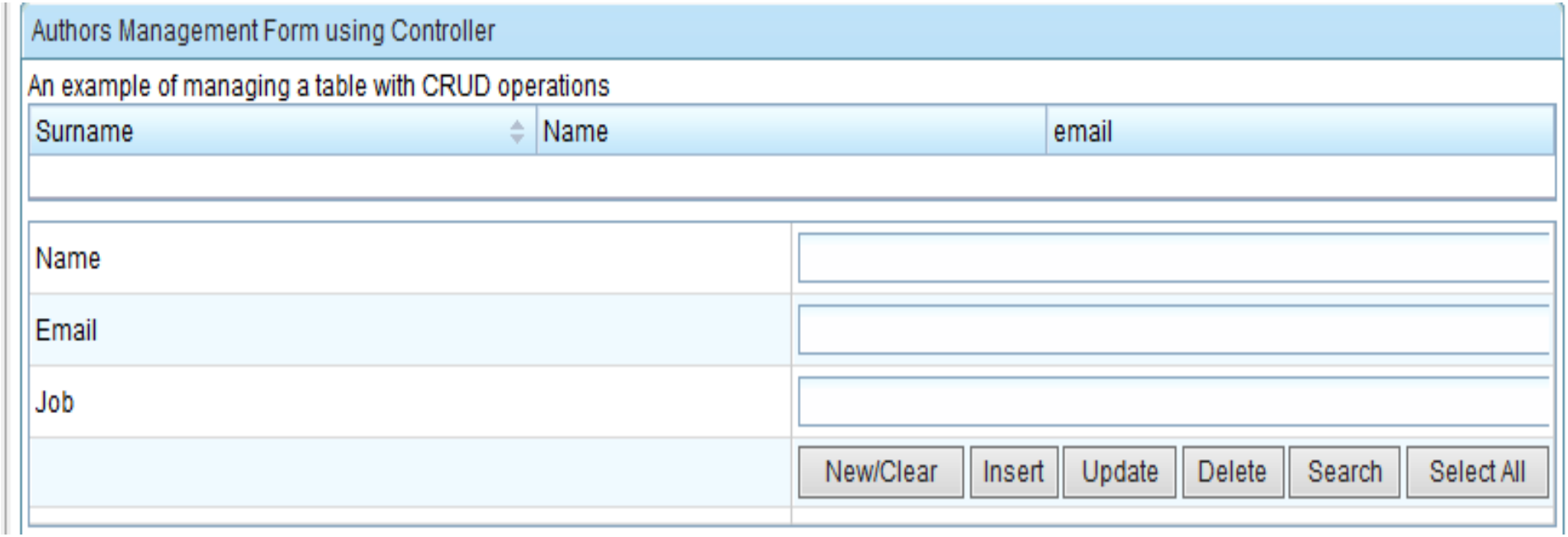
Application Design – Navigation Scheme

- This can be the *web navigation scheme* of our application



Designing Screens

- We will start by defining the Handling screen for each entity, For authors this can be like:



Designing Screens

- Upper: a grid of Authors
- New/Clear: clear form
- Inserts: sends to backend a new author
- Update: updates current
- Delete: deletes current
- Search: applies criteria
- Select All: reselects sets

Designing Screens

- Controls that will be used:
  - Window:
    - Label
    - TextBox
    - listbox (listhead, listheader, listitem,listcell)
    - grid (columns,rows)
    - button
    - Helpful placeholders: vbox, hbox

Implementing Screen Controllers

- Window:
  - Method described here: 1window – 1 page. Could be different...
  - Window implements the logic of a functional unit block: for example handle authors, handle document

Implementing Screen Controllers

- How we programmatically access Window from the Java part:
  - Create a Java sub - class of `GenericForwardComposer`
  - Bind the window in ZUL declaration with your Java sub - class of `GenericForwardComposer` using the `apply` attribute in `window` element.
  - Doing so:
    - You get Java access to all sub controls declared in the window
    - You can trap any events registered with a control of this window

ZK Application

Binding ZUL window to Java controller class

See the example: File author.zul

```
<window
title="Authors Management Form using  Controller"
border="normal"  width="800px"  height="600px"
id="authorWin"
apply="gr.illumine.docreview.web.AuthorFController">
...
</window>
```

Controller implementation class

```
package gr.illumine.docreview.web;

public class AuthorFController extends GenericForwardComposer
implements Serializable{

….

}
```

Each Controller class Must extends GFC!!!

ZK Application

Accessing the Visual Components in the Controller

Declare a component in the zul:

```
<window title="Contact Form"
    apply="gr.illumine.docreview.web.ContactUsFController">
<row>Name<textbox id="nametb"  width="500px" /></row>
<row>Email<textbox id="emailtb"  width="500px" /></row>
<row>Message<textbox id="messagebx"  width="500px" rows="5" cols="40"/></row>
<row><label id="l1"/>
    <hbox>
        <button id="clearbt" label="New/Clear" />
        <button id="sendbt" label="Send" />
    </hbox>
</row>
```

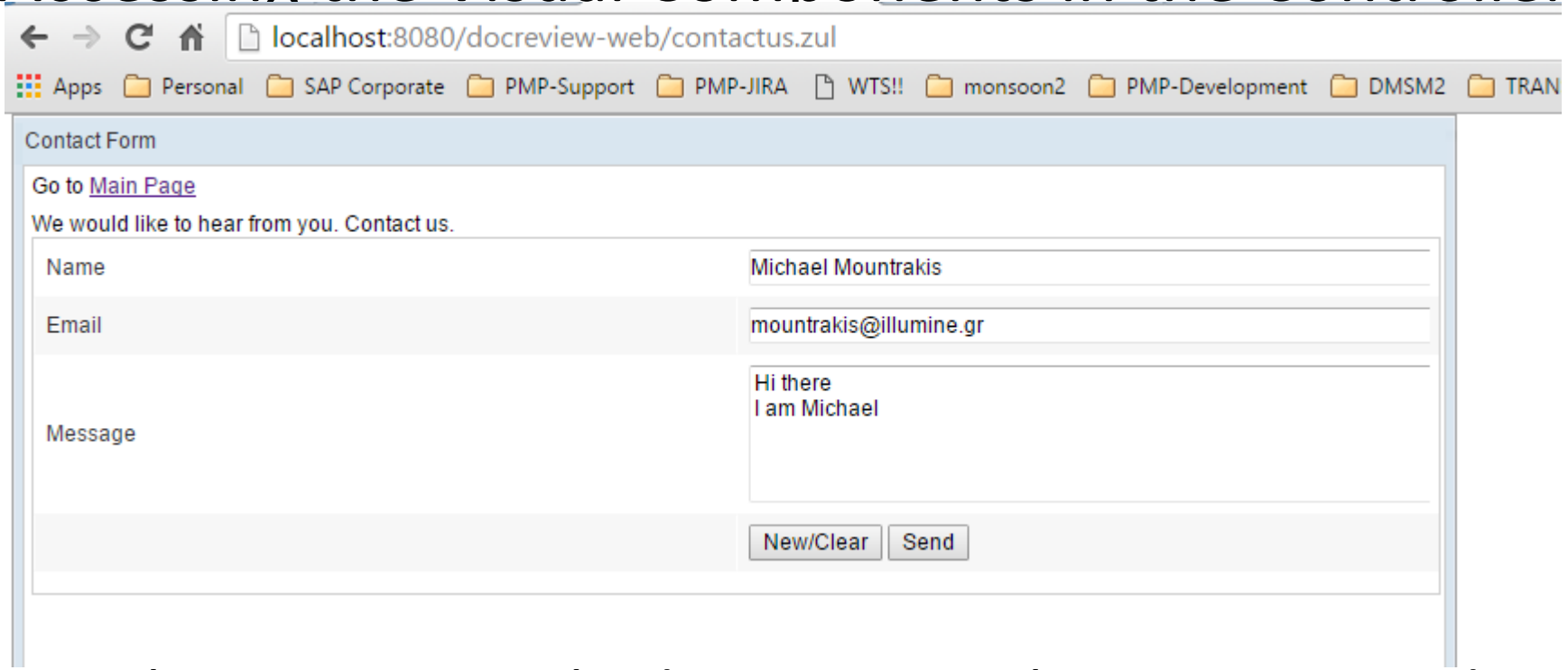
Declare them inside the Controller with their ID as variable name:

```
public class ContactUsFController extends DocReviewGenFC {
    Textbox nametb;
    Textbox emailtb;
    Textbox messagebx;
```

You dont have to declare buttons, unless you want to apply control on them. Instead, use their events.

ZK Application

Accessing the Visual Components in the Controller



Use their access methods to access what user entered:

```
public void onClick$sendbt(){
    debug("onClick$sendbt() Started");
    debug("From : " + nametb.getValue() );
    debug("mail : " + emailtb.getValue() );
    debug("message : " + messagebx.getValue() );
}
```

Multiline text boxes

To upload a file to our application we use a button in the zul:

```
<textbox id="messagebx"  width="500px" rows="5" cols="40"/>
```

And we get the multiline text in the Controller:

```
public void onUpload$uploadBtn( UploadEvent event) {
try {
Media media = event.getMedia();

byte [] bytes = media.getByteData()
String str = media.getStringData();
```

Implementing Screen Controllers SOS

- Populating the listbox from java to zul. Steps
  - Declare the model variable your listbox
  - Greate a getter method for this model variable in your Window Controller Java class
  - Declare the *List Item* that will represent the currentItem of your listbox
  - Declare the mapping of listcell to each attribute of Java Object handled by the *List* variable .

Implementing Screen Controllers SOS

```
<window title="Authors Management Form using  Controller"
        border="normal" width="800px" height="600px"
        id="authorWin"
apply="gr.illumine.docreview.web.AuthorFController">

<listbox id="authorListBox"
         model="@{authorWin$composer.authors}"
         selectedItem="@{selectedAuthor}"
         mold="paging" pageSize="4">

  <listhead sizable="true">
    <listheader label="Surname"  sort="auto"/>
    <listheader label="Name"  />
    <listheader label="email"    />
  </listhead>
  <listitem self="@{each='author'}" value="@{author}">
    <listcell label="@{author.surname}" />
    <listcell label="@{author.name}" />
    <listcell label="@{author.email}" />
  </listitem>
</listbox>
```

Declare Forward Controller class

On Initialization ZK Calls `List<Author> getAuthors()` Of the Controller class.

This is the current iterator Author of `List<Author>` and the mapping of each cell to a property of Author class

ZK Application

Implementing Screen Controllers SOS

```
public class AuthorFController extends GenericForwardComposer {

  private Listbox  authorListBox;

   public AuthorFController() {}

   @Override
   public void doAfterCompose(Component comp) throws Exception {
        super.doAfterCompose(comp);
   }

   public List<Author> getAuthors(){
   List<Author> authorsList = null;
        Try {
            // Use the DB logic to get a  List<Author>
            authorsList = DocReviewLogic.AuthorSelectAll();
        } catch (Exception e) {
            handleError(e, "AuthorFController()");
        }

        return authorsList;
   }
}
```

Implementing Screen Controllers SOS

Make sure:

1) Author has getters/setters for all data members you declare in zul

2) `List<Author> getAuthors()` is implemented in Controller class

3) Your JDBC Driver jar is inside the WebContent/WEB-INF/lib class of your applications WAR deployment or your model JAR is an all inclusive bundle (contains all required jars to connect to the database).

Implementing Screen Controllers SOS

To complete the listbox:

```
selectedItem="@{selectedAuthor}"
        mold="paging" pageSize="4"
```

selectedItem: The name of the variable when a row is selected by cursor

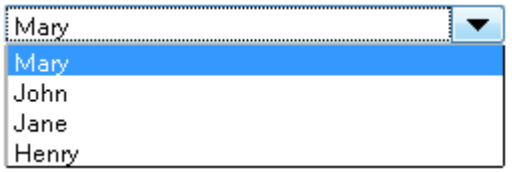
mold: representation of the listbox

pagesize: how many rows to present each time when mold="paging".

mold="default"

name	gender
Mary	FEMALE
John	MALE
Jane	FEMALE
Henry	MALE
This is footer1	This is footer2

mold="select"



mold="paging"

name	gender
Mary	FEMALE
John	MALE
This is footer1	This is footer2

1 /2 [1 - 2 / 4]

ZK Application

Implementing Screen Controllers SOS

To SORT the listbox from one column add the sort attribute :

```
<listhead sizable="true">
    <listheader label="Surname"  sort="auto"/>
    <listheader label="Name"  />
    <listheader label="email" />
</listhead>
```

The Class this listbox stores MUST implement the comparable interface:

```
public class Author implements  Serializable, Comparable<Author>{

    public Author(){
       super();
    }

    @Override
    public int compareTo(Author o) {
     return name.compareTo(o.surname);
    }
```

For Multiple sort options of a list box, follow the directions here:

http://books.zkoss.org/wiki/Small_Talks/2009/January/Multiple_Field_Sorting_on_Listbox

Implementing Screen Controllers SOS

To edit an Item of the listbox we use the following pattern:

- Have a grid of controls that renders all attributes of the Item/Entity
- Each time we pick an Item/Entity from the listbox we bind it ´s attributes to the controls hosted from the grid.
- Grid has an "update" button "update".
- Inside *updateBt$onClick()* we gather updated attributes from the other controls and push the updates to the backend with the use of the Model Logic.

ZK Application

Implementing Screen Controllers SOS

ZUL implementation of the pattern:

selectedAuthor: variable from the Listboxis used in the subsequent grid control to bind infromation

```
<listbox id="authorListBox"
         model="@{authorWin$composer.authors}"
         selectedItem="@{selectedAuthor}"
         mold="paging" pageSize="4">
…...
<grid id="authDetails" width="750px">
<columns></columns>
<rows>
<row>Surname<textbox id="surnamebx" width="500px" value="@{selectedAuthor.surname}"/></row>
<row>Name<textbox id="namebx" width="500px" value="@{selectedAuthor.name}"/></row>
<row>Email<textbox id="emailbx" width="500px" value="@{selectedAuthor.email}"/></row>
<row><label/>
     <hbox>
         <button id="newBtn"    label="New/Clear"/>
         <button id="insertBtn" label="Insert"/>
         <button id="updateBtn" label="Update"/>
         <button id="deleteBtn" label="Delete"/>
         <button id="searchBtn" label="Search"/>
         <button id="selAllBtn" label="Select All"/>
     </hbox>
    </row>
    <row> <label id="dummmy"/> <label id = "resLabel" /> </row>
</rows>
</grid>
```

Uploading files – onUpload event

To upload a file to our application we use a button in the zul:

```
<button id="uploadBtn"  upload="true" label="Upload a file"/>
```

And we handle the onUpload event in the Controller:

```
public void onUpload$uploadBtn( UploadEvent event) {
try {
Media media = event.getMedia();

byte [] bytes = media.getByteData()
String str = media.getStringData();
```

ZK Application

The selected item of the ListBox

To pass the selected Item/Entity from the UI to the FC we do:

```
<window title="Document Management Controller"
        border="normal" width="800px" height="600px"
        id="documentWin"
        apply="gr.illumine.docreview.web.DocumentFControllerEdit">

    Go to  <a href="index.zul" label="Main Page" />
    <listbox id="documentListBox"
             model="@{documentWin$composer.documents}"
             selectedItem="@{selectedDocument}"
             mold="paging" pageSize="4">
```

In the FC Class we do:

```
DocumentExt selectedDocument;
public DocumentExt getSelectedDocument() {
    return selectedDocument;
}
public void onSelect$documentListBox(){
    debug("onSelect$documentListBox() Started");
    selectedDocument=  (DocumentExt)this.documentListBox.getSelectedItem().getValue();
    debug("onSelect$documentListBox() Document Id " +  selectedDocument.getId() );
}
```

Sharing Functionality in Forms

Three basic Patterns – with the difficulty level.

- Encapsulation: Create a self contained component, use it many times. Basic level.
- Inheritance: A base FC implements methods. Children inherit fucntionality. Standard proffesional level.
- Use of Generics: A Generic FC implementing methods. On instanciation Entity class bounds and gets controlled. Difficult – but worths the implementation.

Sharing Functionality in Forms

What pattern do I use?

- All of them! I ´ll show some techniques.

Sharing Functionality in Forms

Encapsulation

- Use it to create custom Components.
- Example Cases it applies:
  - Create a ZUL page along with the FController window to send emails. The *Contact Us* form. *Novice*
  - Create your custom FC Class for user dialogs regarding Error Handling. *Average*
  - Create a dynamic custom window panel with thumpnails. *Difficult*
- Encapsulate the custom controller inside your FC

Sharing Functionality in Forms

Inheritance

- Use it to create *similar* components
- Example Cases it applies:
  - Create a general abstract base FController that implements Error Handling, Clear of child components, Basic data binding... Average
  - Create a custom FController to display a listBox – the child of this controller will implement editable list box. Average
- In general, it is used to assist custom component design.

Sharing Functionality in Forms

Generics

- Use it to create a component ONCE use it with any entity.
- Example Cases it applies:
  - Create a generic FC that displays a list of entities/items in a Grid or a Listbox. Difficult.
  - Create a generic FC that displays a list of editable entities/items in a Grid or a Listbox. Difficult.
- In general, it is used to assist any component design.
- See this one:

https://github.com/eellak/scriptum/blob/master/eProtocol/src/gr/scriptum/eprotocol/controller/GenericEntityController.java

Resources

- ZK Documentation
  - http://www.zkoss.org/documentation
- ZK Forum
  - http://forum.zkoss.org/questions/
- ZK API Documentation
  http://www.zkoss.org/javadoc/

Any Questions?

ZK Application

Lab

- Prepare Workstation
  - Prefereable ORACLE JDK version:1.7.0_79
  - Eclipse IDE for Java EE Developers - Helios
  - Configure Eclipse
    - To run with JDK rather than JRE
  - Setup Application Server: Apache Tomcat 7
    - Bind it with Eclipse
  - Install ZK Studio through Marketplace

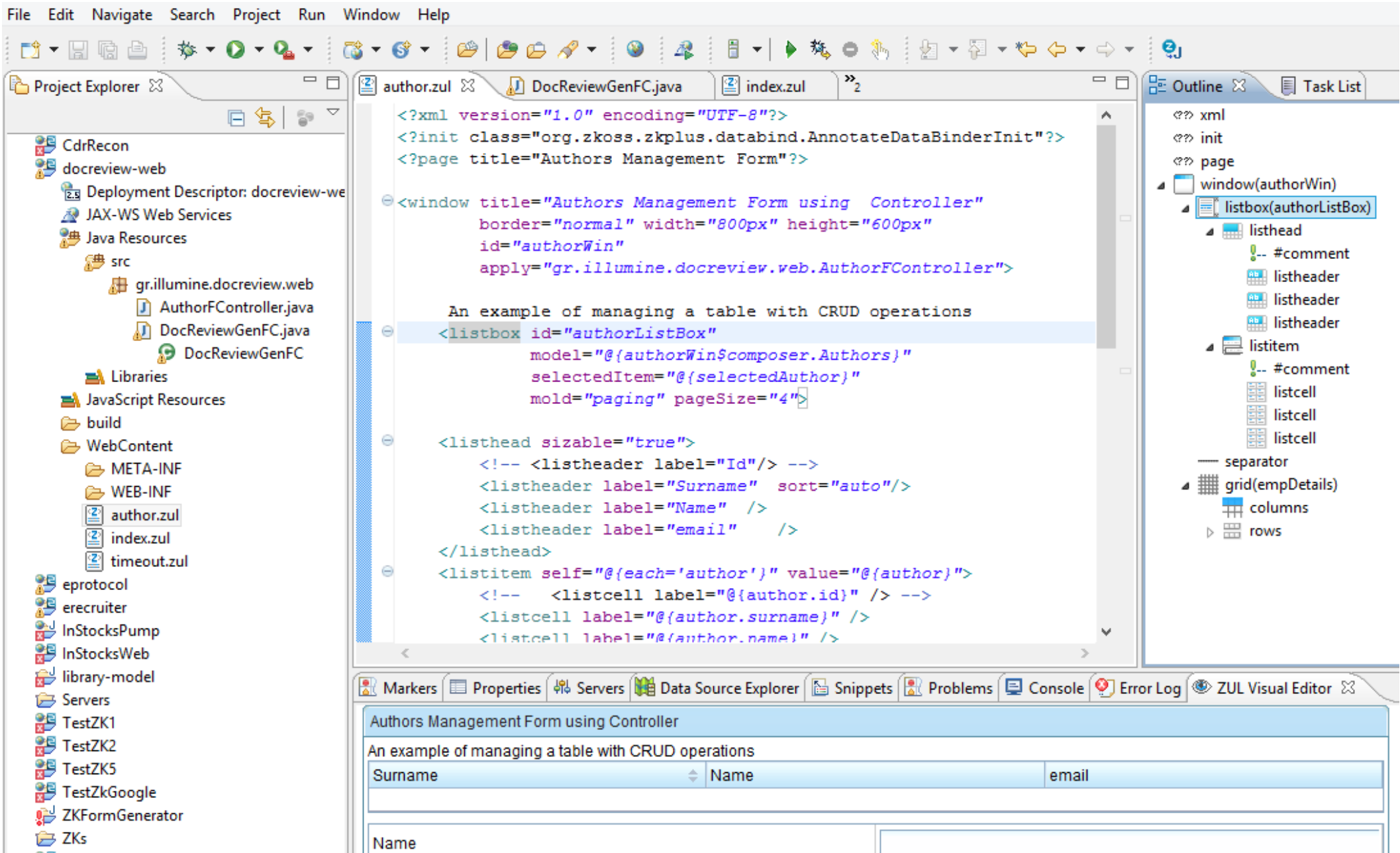
  See:

http://books.zkoss.org/wiki/ZK_Studio_Essentials/Installation

illumineit.com

ZK Application

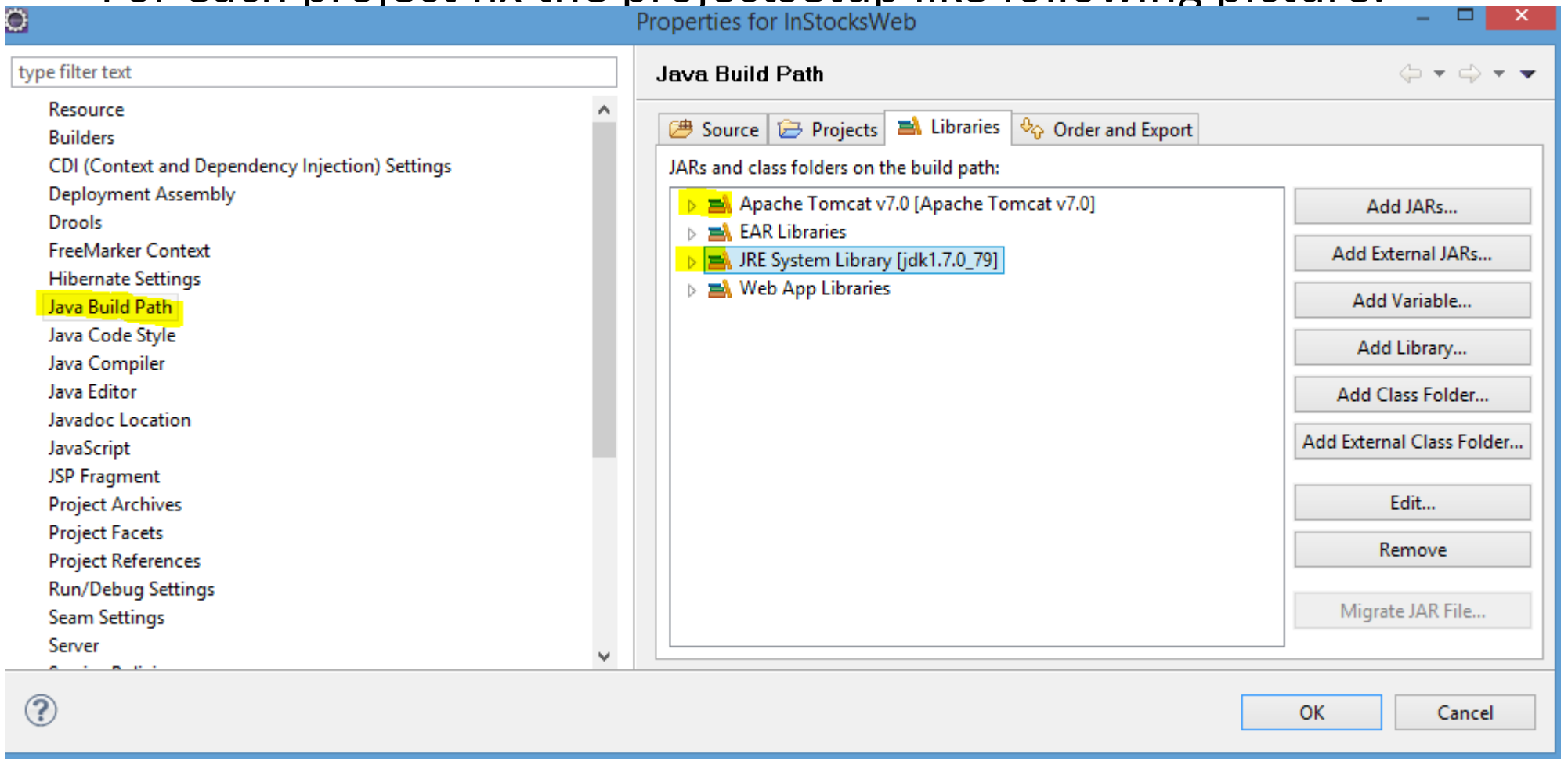
Your Workstation in ZK Perspective



ZK Application

Lab

- Open Eclipse Helios the workspace-zk
- For each project fix the projectsetup like following picture:



ZK Application

Lab

- Excersise1: Implement a simple listbox with a forward controller class just to represent a list. Do so by copying and modifying code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?init class="org.zkoss.zkplus.databind.AnnotateDataBinderInit"?>
<?page title="Authors Management Form"?>

<window title="Authors Management Form using  Controller"
        border="normal" width="800px" height="600px"
        id="authorWin"
        apply="gr.illumine.docreview.web.AuthorFController">

<listbox id="authorListBox"
         model="@{authorWin$composer.authors}"
         selectedItem="@{selectedAuthor}"
         mold="paging" pageSize="4">

<listhead sizable="true">
<!-- <listheader label="Id"/> -->
<listheader label="Surname"  sort="auto"/>
    <listheader label="Name"  />
<listheader label="email"    />
</listhead>
<listitem self="@{each='author'}" value="@{author}">
<!--   <listcell label="@{author.id}" /> -->
<listcell label="@{author.surname}" />
<listcell label="@{author.name}" />
<listcell label="@{author.email}" />
</listitem>
</listbox>
</window>
```

ZK Application

Lab

- The Java Controller part:

```
import gr.illumine.docreview.jdbc.Author;
import gr.illumine.docreview.jdbc.logic.DocReviewLogic;

import java.util.List;
import org.zkoss.zk.ui.Component;
import org.zkoss.zul.ListModel;
import org.zkoss.zul.ListModelList;
import org.zkoss.zul.Listbox;


public class AuthorFController extends DocReviewGenFC {
private static final long serialVersionUID = 1L;

    private Listbox  authorListBox;

    public AuthorFController() {}

    @Override
    public void doAfterCompose(Component comp) throws Exception {
        super.doAfterCompose(comp);
    }
    public List<Author> getAuthors(){
      List<Author> authorsList = null;
       try {
                authorsList = DocReviewLogic.AuthorSelectAll();
            } catch (Exception e) {
                e.printStackTrace();
       }
        return authorsList;
    }
}
```

Lab

- In the previous list box add a sort option.
- Excersise 2: Create and implement a *Contact Us* page.
  - From Name, From email, From message
  - Send message button
  - Clear message button

Lab

- Excersise 3: Create and implement an *About* page.
  - Has your company logo. When you click on it, creates a new browser tab with your company´s URL.
  - Has some textual info about the application
  - Has a list of developers with their names and emails
  - Has a list of useful links

Lab

- Excersise 4: Create and implement an simple window with one Upload Button and a Blank image.
  - When upload is pressed, a dialog appears to load an image
  - Acceptable images: accept only PNG, JPG, GIF of a size less than 64kb
  - Warn user with a dialog if image is corrupted.

Lab

- Excersise 5: Create and implement an simple window with an editable list box.
  - Entity has a picture, a name, phone and email
  - Create, update, delete, search operations
  - Acceptable images: accept only PNG, JPG, GIF of a size less than 64kb
  - Warn user with a dialog if image is corrupted.
  - Use a user dialog to Create/Update entity (modal view)