# Component Based Approach Wrap Up

JSF Slides #4

# Content

- JSF and AJAX
- Fileupload
- HTML5 and JSF
- Application Layers
- MAter detail interface
- Clean URLs
- Pagination
- JEE Authorization

# AJAX Behaviour

## The AJAX element

```
<h:form>
    <h:inputText value="#{ajaxbean.bound}" />
    <h:commandButton ... >
        <f:ajax render="output" execute="@form" />
    </h:commandButton>
    <h:outputText id="output" value="#{ajaxbean.randnumb}" />
</h:form>
```

JSF is default “full page load”

- Suitable for many application
- Need for AJAX ...?? If so …
  - Many component frameworks have components with default AJAX behaviour, more later …
  - Worst case: Handle it self, using <f:ajax>-element

Tech talk

- The <f:ajax> requires jsf.js (library supplied by JSF) file
  - Must have <h:head> element to trigger download
- <f:ajax> inside <h:form>
- Enclos or nest element(s) with <f:ajax> to give AJAX behaviour
- Uses id-attribute of elements to specify input and output elements
- Attributes
  - render: The elements to redisplay on the page.. Often h:outputText
    - The target of the render must be inside the same h:form as <f:ajax>
  - execute: The element(s) to send to server to process.
    - Generally input elements such as h:inputText or h:selectOneMenu.
  - event: The DOM event to respond to (e.g., keyup, blur)
  - … and more

  There are 4 special values for attributes execute and render
  - @this. The element enclosing f:ajax.
  - @form. The h:form enclosing f:ajax.

    - Very convenient if you have multiple fields to send
- @none. Nothing sent.
    - Useful if the element you render changes values each time you evaluate it.
- @all. All JSF UI elements on page.

# Fileupload

## Using inputFile

```
<h:form ...  enctype="multipart/form-data">
    <h:inputFile ... value="#{uploadbean.file}"
      validator="#{uploadbean.validateFile}"/>
</h:form>
```

There's an element for fileupload

# HTML5 and JSF

## Pass through elements

```
<input type="email" placeholder="Enter email"
    jsf:value="#{htmlbean.email}" required="required"/>

<label jsf:value="#{htmlbean.email}" />
```

## Pass through attributes

```
<h:inputText id="email" value="#{htmlbean.email}"
        p:type="email" p:placeholder="Enter email"/>
```

Possible to let JSP and HTML work in conjunction (HTML5 friendly JSF)

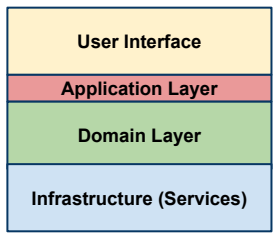
Pass-through elements

- In HTML5 (i.e. no JSF-elements) use JSF attributes (and EL expressions)
- Make it possible to build the view with HTML elements that can access components and beans
  - Will convert HTML to JSF
  - Mapping between HTML-elements and JSF in TagDecorator
- To make an element a pass-through element, at least one of its attributes must be in the namespace http://xmlns.jcp.org/jsf.

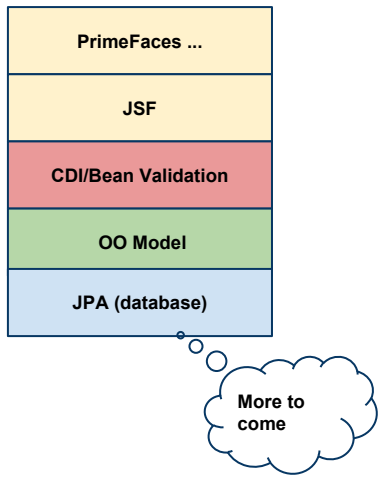
Pass-through attributes

- In JSF (using JSF-elements), use HTML5 attributes
- Attributes passed through JSF directly to render phase
- Prefix attributes with namespace http://xmlns.jcp.org/jsf/passthrough

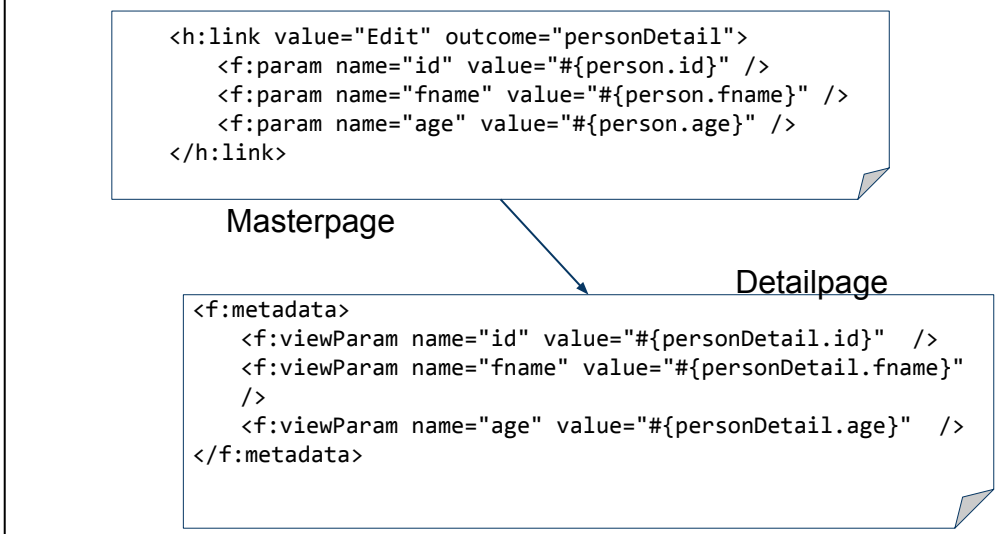
# Application Layers




Domain driven
application layering



# Master Detail Interface




Data from master page sent to page beans for detail page

# Clean URLs and Pagination


(1 of 5) 1 2 3 4 5 10

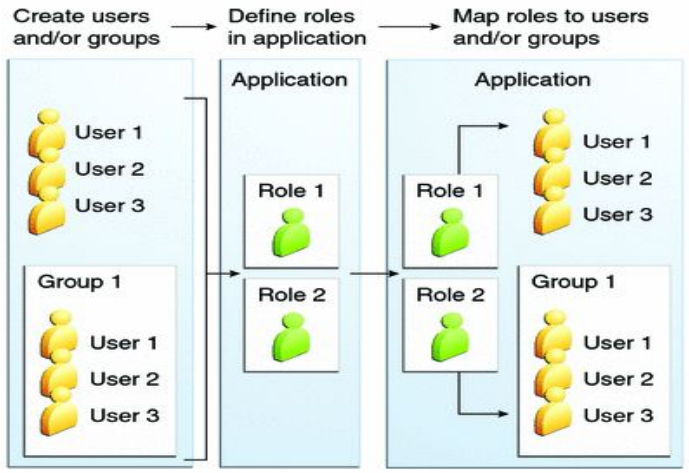
| Id | Year | Brand | Color |
|---|---|---|---|
| 98baf50f | 1963 | Honda | Black |
| 00ffb1b8 | 1990 | Fiat | Red |
| 4857916a | 1984 | Honda | Green |
| b430b90b | 1969 | Ford | Yellow |
| fd7936d0 | 1988 | Jaguar | Black |
| b74c09da | 1974 | BMW | Blue |
| 13248975 | 2006 | Jaguar | Blue |
| 0415b645 | 1960 | Fiat | Maroon |
| fcd5b38a | 1969 | Volkswagen | Silver |
| 6d3f1579 | 1970 | BMW | Green |

(1 of 5) 1 2 3 4 5 10

No standard (built in solutions for)

- Clean URls-
  - Third party solution from OCPsoft
- Pagination
  - Do it yourself or find a component

# JEE Authorization




JEE supports

- Basic authentication
- Form-based authentication
- .. and more

Now we'll use Form-Based (preferred way to do it)

Standard JEE authorization technique, using realms

- A realm is a security policy domain defined for a web or application server. A realm contains a collection of users, who may or may not be assigned to a group

Types of realms (supported by GlassFish and Tomcat)

- file, Stores user information in a file. This is the default realm when you first install the GlassFish Server
- ldap, Stores user information in an LDAP directory
- jdbc, Stores user information in a database
- For now we use the file realm with GlassFish
  - This is server dependent (Tomcat different)
  - Database backed sample later (better)

Terms

- A role is an abstract name for the permission to access a particular set of resources in an application.
- Web resource collection

    - A list of URL patterns (the part of a URL after the hostname and port you want to constrain) and HTTP operations (the methods within the files that match the URL pattern you want to constrain) that describe a set of resources to be protected.
- Authorization constraint
    - Specifies whether authentication is to be used and names the roles authorized to perform the constrained requests.
- User data constraint
    - Specifies how data is protected when transported between a client and a server.

Steps

- Create users and groups in GlassFish file realm (using Admin console) …
- Create roles in application (defined in glassfish-web.xml)
- Map roles to users and groups in web.xml
- Specify security constraints in web.xml