Oncore CHHS User Interface Guide

Introduction

The CHHS demonstration website is one example of OnCore's expertise in constructing rich enterprise web applications. The Foster Care demonstration website is constructed from top to bottom in open source styles, components, development tools, and server technologies. The OnCore team chose to make use of readily accessible responsive CSS templates freely available under the Creative Commons license for two reasons. First, a conscious decision was made to ensure the look and feel of the site did not resemble any existing California government or Federal government website as it is publicly accessible. There is a real concern the site could be visited by the public and confused with a real application. Secondly, due to the aggressive schedule for completing the application and its open source nature, we determined it critical to reuse readily available open source components to the greatest extent possible.

The OnCore team built the application on the tried and proven Java Enterprise Edition (JEE7) platform using Java Server Faces (JSF) for page construction. The PrimeFaces extension of the core JSF component library was chosen as the primary component set due to OnCore's previous experience implementing PrimeFaces on successful projects. PrimeFaces, which builds on the core Java Server Faces components, provides an incredibly rich set of tools for building robust and rich AJAX web applications.

The Java Server Faces architecture provides excellent separation between the User Interface (UI), business logic, and data layers of the application. In addition, JSF has support for creating common templates, shared fragments, and reusable components, making page construction quick and relatively straight forward. Further, JSF's ability to easily integrate with existing HTML prototypes, means pages can be mocked up in HTML design tools or the designer can use off the shelf HTML templates and easily convert the HTML into JSF. Combine this with the excellent PrimeFaces component library and just about any application scenario can be covered.

PrimeFaces

PrimeFaces is an open source extension of the stock JSF implementation providing an extensive set of rich components, including:

* Rich set of components (Html Editor, Dialog, AutoComplete, Charts and many more).

* Built-in Ajax based on standard JSF 2.0 Ajax APIs.

* Lightweight, one jar, zero-configuration and no required dependencies.

* Push support via Atmosphere Framework.

* Mobile UI kit to create mobile web applications.

* Skinning Framework with 35+ built-in themes and support for visual theme designer tool.

* Extensive documentation.

* Large, vibrant and active user community.

* Developed with "passion" from application developers to application developers.

PrimeFaces provides a set of pre-packaged templates that cover most scenarios. The OnCore team chose to use the twitter bootstrap theme for the OnCore CHHS demo as it complemented the color pallet and font schemes of the chosen HTML template. However, changing themes is as simple as changing one setting in the web.xml file of the application. Designers can also create their own custom themes, , for the demo application we chose to customize the already available bootstrap theme package.

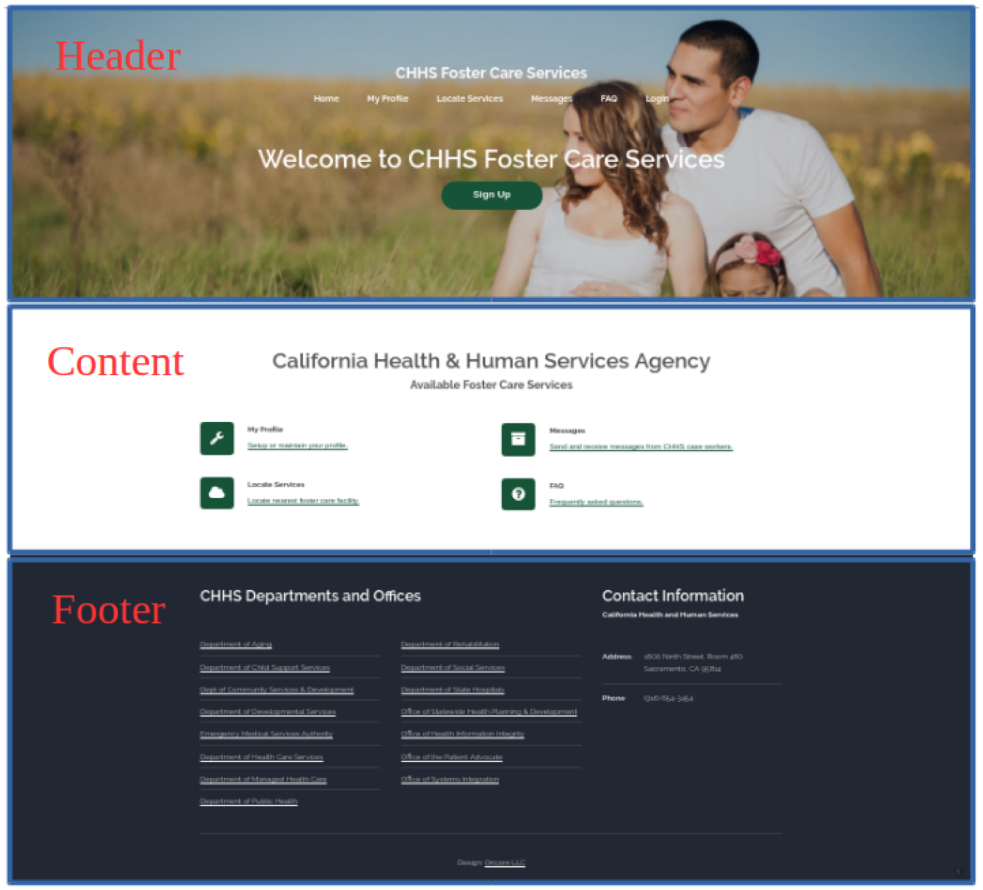
For more on PrimeFaces visit the PrimeFaces website.

Templates

JSF provides the ability to create page templates, which consolidate common elements into one location. Using a template, common page structure, shared CSS, JavaScript, and other elements can be placed in one file and then extended by other JSF pages in a similar manner to a class extending a base class. This allows for easier page construction and maintenance as common elements can be changed in one location and propagated across the entire application.

The OnCore CHHS demo site makes extensive use of templates to simplify page construction and encourage code reuse. The base template is the common_layout.xhtml JSF file.

Common Layout Template


Header
CHHS Foster Care Services
Home
My Profile
Locate Services
Messages
FAQ
Login
Welcome to CHHS Foster Care Services
Sign Up
Content
California Health & Human Services Agency
Available Foster Care Services
My Profile
Setup or maintain your profile.
Messages
Send and receive messages from CHHS case workers.
Locate Services
Locate nearest foster care facility.
FAQ
Frequently asked questions.
Footer
CHHS Departments and Offices
Department of Aging
Department of Child Support Services
Dept of Community Services & Development
Department of Developmental Services
Emergency Medical Services Authority
Department of Health Care Services
Department of Managed Health Care
Department of Public Health
Department of Rehabilitation
Department of Social Services
Department of State Hospitals
Office of Statewide Health Planning & Development
Office of Health Information Integrity
Office of the Patient Advocate
Office of Systems Integration
Contact Information
California Health and Human Services
Address
1600 Ninth Street, Room 460
Sacramento, CA 95814
Phone
(916) 654-3454
Design: Oncore LLC
Exhibit 1: Oncore CHHS Demonstration Common Layout

The common_layout.xhtml template is the foundation for all other templates and pages within the OnCore CHHS application. It defines the basic structure of all the pages and is the place to drop in common elements such as CSS style sheets and JavaScript. As seen in the above screen shot, the page structure is a basic header, content, footer scheme. JSF provides an easy mechanism for defining these structures and extending or implementing them in other templates or pages respectively. This is accomplished using the "<ui:insert>" tag. Let's look at the basic structure of the common_layout.xhtml template.

```
    <h:head>
    ... import style sheets etc.

  </h:head>


  <h:body styleClass="homepage">

    <f:view>

      <h:form id="chssForm">


        <ui:insert name="header">

          <!--  INSERT HEADER HERE -->

        </ui:insert>


        <ui:insert name="content">

          <!-- INSERT CONTENT HERE -->

        </ui:insert>


        <ui:insert name="footer">

          <!-- INCLUDE FOOTER HERE -->

        </ui:insert>

      </h:form>

    </f:view>

  </h:body>

</html>
```

The "<ui:insert>" tags are used to specify portions of the file that can be overwritten or extended by implementing pages. This is accomplished by using the "<ui:define>" tag as seen in the following example taken from the OnCore CHHS application main_layout.xhtml, which extends common_layout.xhtml.

```
<ui:composition .....

        template="common_layout.xhtml">




  <ui:define name="header">

    <ui:include src="../common/header.xhtml"/>
  </ui:define>



  <ui:define name="footer">
    <ui:include src="../common/footer.xhtml"/>
  </ui:define>


</ui:composition>
```

In this example, a new template is defined which implements the common layout template, overriding the header and footer with specific content.

Looking at the main index page of the site, we can see that the page only need implement a template and override the content section. This is the pattern for all the pages in the site and should be adhered too for further page development.

```
<ui:composition template="/facelets/templates/main_layout.xhtml">


        <ui:param name="managedBean" value="#{homeManagedBean}"/>


        <ui:define name="title">
            <title>California Health &amp; Human Services | Foster Care Services</title>
        </ui:define>


        <ui:define name="content">


            <!-- Main -->
            <div id="main" class="wrapper style1">
                <section class="container">


                    .......
        </ui:define>


    </ui:composition>
```

The following diagram demonstrates how the pages are constructed using JSF templating previously described above. If a new page is needed, the page would use the reduced_layout.xhtml template as the foundation as noted in the diagram. The reduced_layout template alters the header, changing the image, removes the sign up button, and reduces the vertical height of the image.

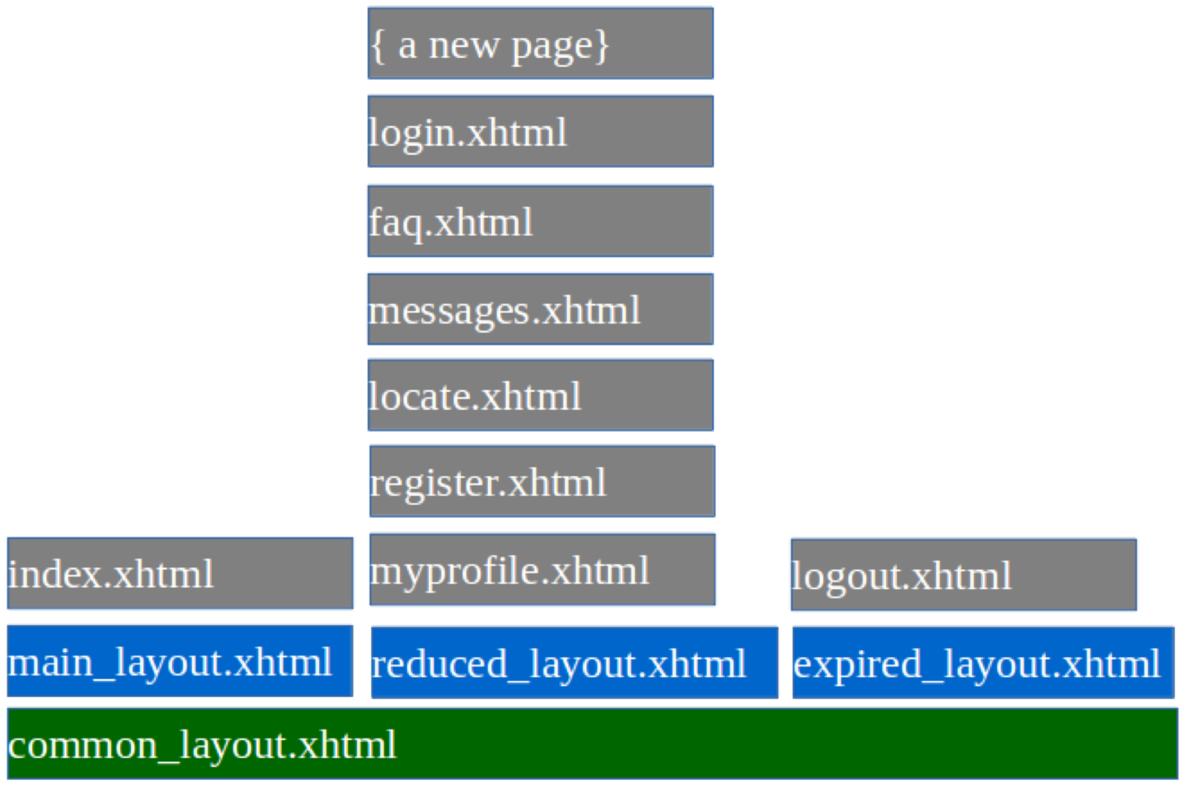


Exhibit 2: OnCore CHHS Demonstration Page Construction

Facelets

In addition to templates, JSF provides an incredibly easy to use mechanism for code reuse called facelets. Think of facelets as reusable code blocks, which can be used to build out pages. Using the "<ui:define>" mechanism mentioned in previous sections developers can easily insert different versions of code based on any criteria. For example, it is common to have different headers for a website driven by factors such as role or authentication. Using facelets the developer can easily create different headers for different situations and inject them into the page by using the define directive.

For example, in the main_layout.xhtml template, the template is importing a header and footer as follows.

```
<ui:composition .....
        template="common_layout.xhtml">



  <ui:define name="header">
    <ui:include src="../common/header.xhtml"/>
  </ui:define>


….. etc


The header facelet contents looks something like this..
 <ui:composition>


    <!-- Header -->
    <div id="header">


      <div class="container">


        <a class="skip-main" href="#main">Skip to main content</a>
        <!-- Logo -->
        <h1><a href="index.xhtml" id="logo">CHHS Foster Care Services</a></h1>


        <!-- Nav -->
        <nav id="nav">
          <ul>
```

```
            <li><b><a href="index.xhtml">Home</a></b></li>
            ……. etc
</ui:composition>
```

Components

JSF provides another powerful mechanism for facilitating code reuse in the form of components. Components are similar to facelets except they expose properties. In the OnCore CHHS demo application, the component set is limited to input text, mask, and drop downs as the functionality is limited. However, there is no limit to the number and variety of components that can be created. Generally a component will encapsulate a stock JSF or PrimeFaces component and extending its capabilities or providing further structure around the component. This greatly reduces the amount of code needed for each page.

For example, take the standard input text control commonly used on most pages. The following code block is typical of an input text box. If the page has 20 input text fields then there would be a considerable amount of code on the page; if it is decided at a later date to change the formatting of the fields by let’s say, adding a message to the right of each input text box, then the change would be extensive.

```
<p:panelGrid id="test_pnl" columns="2" columnClasses="column_1_class,column_2_class" role="presentation">


        <p:outputLabel for="input" styleClass="somestyle">
            <h:outputText value="somelabel" rendered="#{!someManagedBean.isRequired}"/>
            <h:outputText value="* somelabel" rendered="#{someManagedBean.isRequired}"/>
         </p:outputLabel>


         <p:inputText
            id="input"
            rendered="#{someManagedBean.isRendered}"
            value="#{someManagedBean.value}"
            autocomplete="off"
            immediate="true"
            maxlength="50"
            readonly="false"
            size="30"
            title="some title text"
            styleClass="some style"
           />
```

```
            <h:outputText/>


            <p:message for="input" showDetail="false" showSummary="true" display="text"/>


</p:panelGrid>
```

Now let's look at how this would look if a component is used instead. As can be seen in the example below, the formatting has been abstracted away from the developer. The UI developer can place the component on the page, modify the properties, and move on without the need to worry about additional formatting. Further, now if a decision is made to modify the input text fields in the application, that change can take place in one place and propagate throughout the application.

```
<onc:inputText_1 id="userNameTxt"

      label="User Name:"

      labelStyleClass="form_label"

      inputStyleClass="input_float_left"

      requiredLabel="true"

      maxlength="45"

      size="25"

      immediate="true"

      errAlt=""

      errMsgRendered="false"

      title="#{managedBeanContent.userNameError}"

      value="#{managedBeanContent.userName}"/>
```

When constructing new pages in the OnCore CHHS reference application, use the following components:

Component	Description	When to Use
inputText	Wraps the PrimeFaces inputText component	Use for standard text input fields
inputMask	Wraps the PrimeFaces inputMask component	Use for masked input text fields
InputTextArea	Wraps the PrimeFaces inputMask component	Use for text area input fields

Business Tier (Providing Business Logic Support)

The Java Server Faces architecture provides excellent separation between the user interface layer of the application, the business tier, and the data tier. The Oncore CHHS prototype follows a pattern of one to one support between page, business tier object, and data tier object. Following a set pattern helps keep page construction simple and repeatable, improving development time and maintenance.

Using the register new user page, register.xhtml, as an example the page and supporting class structure looks like the following:

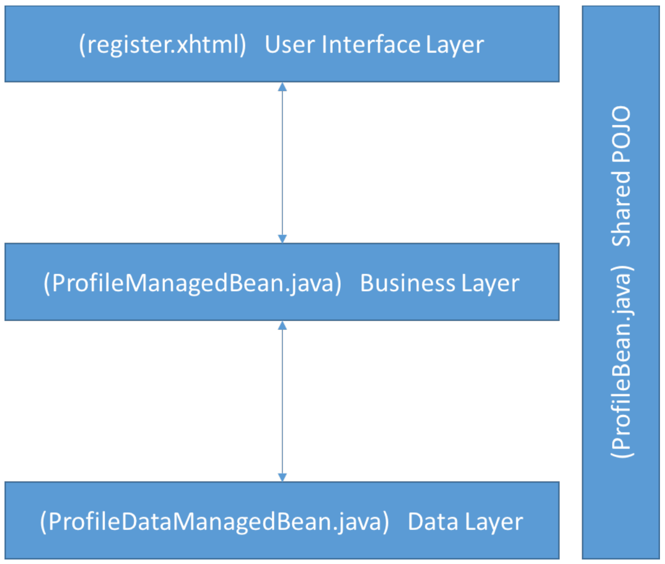

(register.xhtml) User Interface Layer
(ProfileManagedBean.java) Business Layer
(ProfileDataManagedBean.java) Data Layer
(ProfileBean.java) Shared POJO

Exhibit 3: Register New User Page Class Structure

The binding between business object and page is accomplished using the Context Dependency Injection or CDI. CDI allows JSF pages to interact with a supporting Java object by using the "#{cdi name}" tag. CDI is also used at the business and data tiers to allow Java classes to inject fully initialized copies of other Java objects using the @Inject annotation. CDI further provides other benefits, such as the ability to create alternative implementations of classes and control which implementation is used at runtime by specifying the class in the beans.xml file. This helps with page construction and testing as mock objects can be created to simulate the final objects, allowing coding to progress even when a layer is not complete.

The register.xhtml page uses the param attribute to define two aliases to the managed beans supporting the page.

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

    xmlns:h="http://java.sun.com/jsf/html"

    xmlns:ui="http://java.sun.com/jsf/facelets"

    xmlns:c="http://java.sun.com/jsp/jstl/core"

    xmlns:f="http://java.sun.com/jsf/core"

    xmlns:onc="http://java.sun.com/jsf/composite/onc"

    xmlns:p="http://primefaces.org/ui">


  <ui:composition template="/facelets/templates/reduced_layout.xhtml">

     <ui:param name="managedBean" value="#{profileManagedBean}"/>

     <ui:param name="managedBeanContent" value="#{profileManagedBean.profileBean}"/>


……

<onc:inputText_1 id="userNameTxt"

…..

      title="#{managedBeanContent.userNameError}"

       value="#{managedBeanContent.userName}"/>


<p:commandButton id="registerBtn" value="Sign Up"

      styleClass="buttonGroup" onstart="PF('feedbackWdg').show()"

      oncomplete="PF('feedbackWdg').hide()"

      ajax="false"

      action="#{managedBean.handleRegisterButtonClickEvent()}"/>
```

Here we see the page is creating two aliases, "managedBean" pointing to the ProfileManagedBean.java class and "managedBeanContent" pointing to an instance of the ProfileBean.java class referenced as property of ProfileManagedBean. The input text control uses the managedBeanContent alias to access the "userName" property. The PrimeFaces command button uses the action property to invoke the "handleRegisterButtonClickEvent" method on the ProfileManagedBean.

It is important to note the use of the <ui:param> tag to create an alias is not required. We could have just used the object directly, however, doing so would make the code less reusable. For example, we might have two pages that share the same data input fields, but supported by different business layer managed beans. Using an alias allows one facelet to be shared on both pages reducing the amount of code necessary to implement the pages.

Looking at the above example, the page refers to the managed bean by a name. In this case, it is “profileManagedBean”. The CDI name is set in the Java class using the @Named annotation. By adding the @Named annotation to a class it is now CDI managed. The other sections of most importance in this example are highlighted in red in the code “snippet” taken from the ProfileManagedBean below.

- @Named: Designates this class as CDI managed and defines the name by which other classes and JSF pages can refer to this class
- @ViewScoped: Designates this class and the page(s) it supports as maintaining scope only while the page is being viewed.
- @PostConstruct: A method annotated with PostConstruct will be called after the object has been constructed. This is a good location to initialize variables and load data for the page.
- @PreDestroy: A method annotated with PreDestroy will be called immediately prior to the class being designated ready for garbage collection. In other words it is no longer used. In the case of using ViewScoped, this method will be called as soon as the user moves away from the page supported by this managed bean.
- Event handlers: In this example there is an event handler designated to handle the user clicking the register button on the page. In most managed beans the business logic is most commonly used for the following purposes, initialization (loading data), event handling, page validation, and persisting data.
- Getters and Setters for ProfileBean. This defines a property for the ProfileBean class. The ProfileBean.java class is a simple bean holding properties used by the various Java classes and pages supporting profiles.
- @Inject: This is a CDI annotation and allows injecting of fully initialized objects into another.

```
/**
 *
 * @author oncore
 */
@Named("profileManagedBean")
@ViewScoped
public class ProfileManagedBean extends BaseManagedBean {
    @Override
    @PostConstruct
    public void initialize() {
        LOG.debug("Initializing ProfileManagedBean: " + this.getClass().hashCode());
    }

    @Override
    @PreDestroy
    public void destroy() {
        LOG.debug("Destroying ProfileManagedBean: " + this.getClass().hashCode());
    }

 /**
   * The <code>handleRegisterButtonClickEvent</code> method handles the click
   * event from the register button on the register screen.
   *
   * @return null if there is an exception, index if the operation is
   * successful
   */
public String handleRegisterButtonClickEvent() {
                …..
}

/**
  * @return the profileBean
  */
public ProfileBean getProfileBean() {
       return profileBean;
 }

/**
  * @param profileBean the profileBean to set
  */
public void setProfileBean(ProfileBean profileBean) {
        this.profileBean = profileBean;
}

@Inject
```

```
AbstractProfileDataManagedBean profileDataManagedBean;


@Inject
AbstractLoginDataManagedBean loginDataManagedBean;


@Inject
 ProfileValidationBean profileValidationBean;


 private ProfileBean profileBean = new ProfileBean();
…..
}
```

Data Tier (Proving Data Support)

Most pages require some type of data support. Either the page needs to fetch data from some underlying data source or the page needs to write data to an underlying data source. There are two data abstraction layers in the OnCore CHHS prototype. The top layer is the data layer directly supporting the user interface layer as described in this document. The second layer is the low level web services and supporting EJB's, which are typically hosted in the middle tier. The middle tier is not covered in this document. The Data Tier from the user interface perspective, provides data services support by abstracting the application from the underlying data source. The underlying data source can be a web service, a database, or a mock layer. In the ProfileManagedBean code snippet shown above the data layer objects are injected directly into the class using the @Inject CDI annotation. Again the injection knows how to resolve the class based on the name given in the @Named annotation.

```
/**
 * This class invokes RESTful services to find, create and update profiles.
 *
 * @author oncore
 */
@Named("profileDataManagedBean")
@RequestScoped
public class ProfileDataManagedBean extends BaseManagedBean implements AbstractProfileDataManagedBean {


@Override
```

```
@PostConstruct
public void initialize() {
       LOG.debug("Initializing ProfileDataManagedBean: " + this.getClass().hashCode());
}


@Override
@PreDestroy
 public void destroy() {
       LOG.debug("Destroying ProfileDataManagedBean: " + this.getClass().hashCode());
 }


 /**
   * Finds the profile information for the user using the userUid.
   *
   * @param userUid
   *
   * @return <code>Profile</code>
   *
   * @throws WebServiceException
   */
@Override
public Profile findProfileByUserUid(Integer userUid) throws WebServiceException {
      return this.getProfileServiceClient().findProfileByUserUid(userUid);
 }


……
```

Styles

The core styles for the application are based on a responsive template provided under the Creative Commons license. However, modifications to the provided templates were made as needed to achieve the desired look and feel.

The stock style sheets are

- font-awesome.min.css
- skel.css
- style-moble.css
- style-narrow.css
- style-narrower.css
- style-normal.css
- style-wide.css
- style.css

Out of these the primary style sheets are style.css and skel.css, which provide the overall structure and define styles for the common HTML elements, fonts, and colors.

In addition, OnCore created two additional style sheets designed for use with the JSF components mentioned in the Components section above as well as customizations to the PrimeFaces theme.

- style-components.css
- style-primefaces_overrides.css

The responsive aspects are handled by a combination of JavaScript and CSS. In addition, the CSS media tag in combination with the JSF facelets facility is used to control data table display per screen size. For mobile screens, data tables with a reduced number of columns is displayed as compared to desktop views.

Constructing a New Page

As the style sheets and supporting JavaScript are already included in the common_layout template file, the task of constructing pages is greatly simplified. The page structure and components are already defined and building additional pages can be quickly accomplished by copying an existing page and modifying the content section.

As an example, the client has requested a new page to allow the foster care parent to search for approved foster care counselors nearest to their current location.

The new page seems very similar to the page allowing foster care parents to search for foster care facilities. Since there is already a search page written for the application, it is desirable to follow the same pattern. Using consistent patterns throughout a user interface is critical to a good user experience.

Steps

1. Copy the locate.xhtml page to {newpage}.xhtml
2. Copy the supporting managed bean SearchManagedBean.java to {NewManagedBean}.java
3. Copy the supporting POJO SearchBean.java to {NewBean}.java
4. Copy the supporting data layer bean(s) SearchDataManagedBean.java AbstractSearchDataManagedBean.java to {NewDataManagedBean}.java and {AbstractNewDataManagedBean}.java respectively.
5. In the locate.xhtml file change the managedBean and managedBeanContent ui:params to point to {NewManagedBean} and {NewManagedBean.NewBean} respectively.
6. In the locate.xhtml file alter the code between the <ui:define name=”title” and <ui:define name=”content” according to the page requirements.
7. Alter the contents of the {New..} classes according to the page requirements.

Summary

The OnCore CHHS demonstration application builds on the excellent foundation of the Java Enterprise Edition platform and supporting technologies from the Open Source Community. The Java Enterprise platform, combined with the open source PrimeFaces extensions to the Java Server Faces component set, provides a rich medium to construct enterprise class applications. Further, using free off the shelf components the OnCore team has demonstrated how to quickly and efficiently build a powerful website meeting current usability, responsive design, and N-Tier architecture standards.