# Oncore CHHS User Interface Guide

## Introduction

The CHHS demonstration website builds on Oncore LLC'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 due to the application being 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 as much as 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 JSF component library was chosen as the primary component set due to Oncore's previous experience implementing PrimeFaces on successful projects. PrimeFaces provides an incredibly rich set of components for building robust and rich AJAX web applications.

JSF provides excellent separation between the 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.

* Rich set of components (HtmlEditor, 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, however, 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

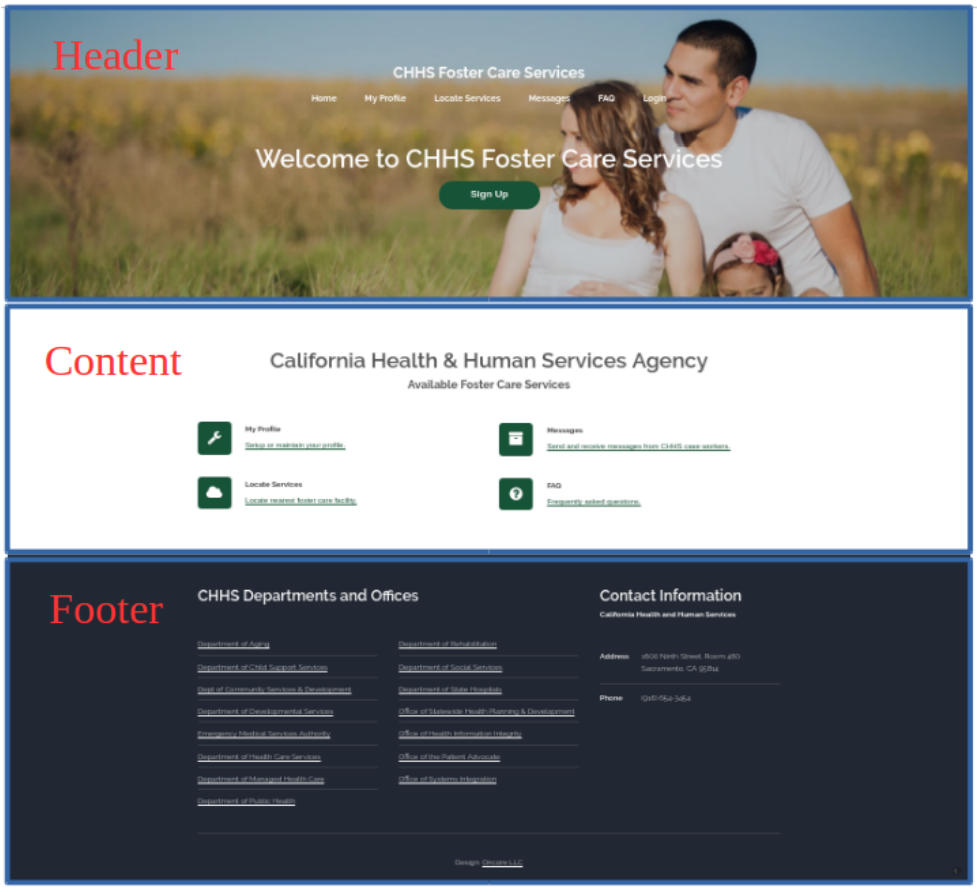


*Illustration 1: Oncore CHHS Demonstration Common Layout*

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;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.

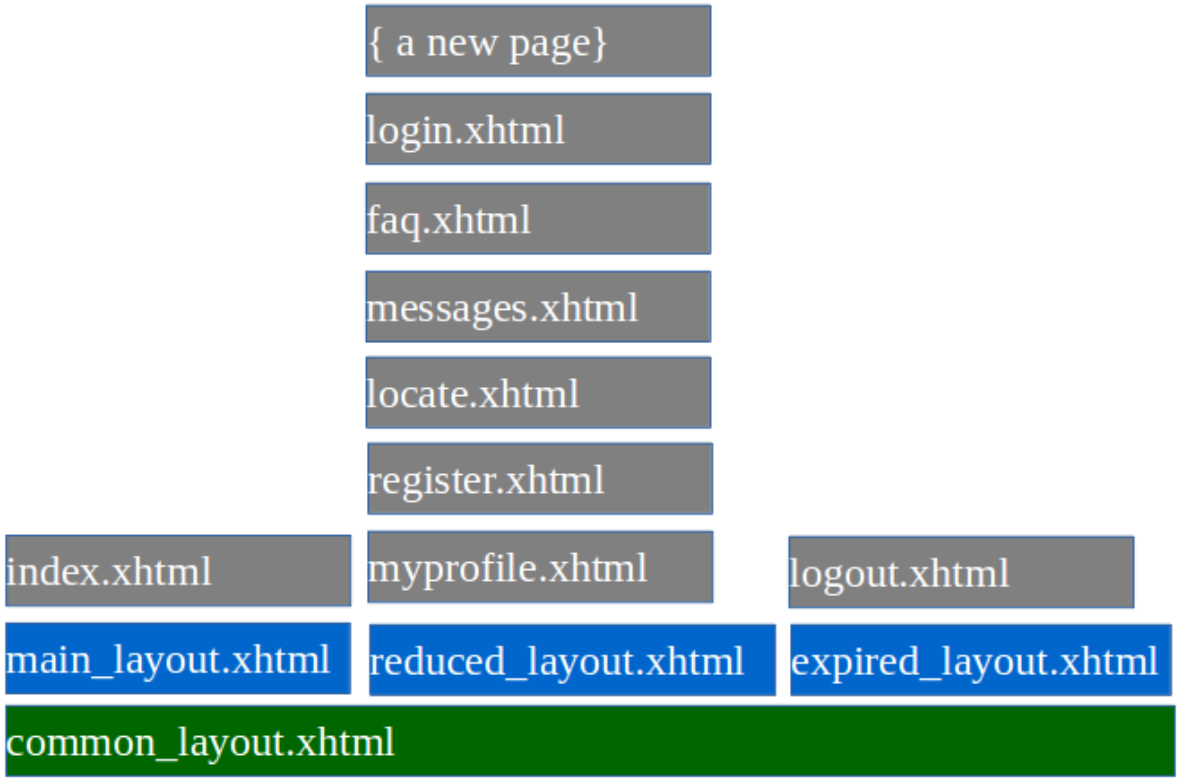


*Illustration 2: Oncore CHHS Demonstration Page Construction*

# 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. Obviously if the page has 20 input text fields then there would be a considerable amount of code on the page. Further, if it is decided at a later date to change the formatting of the fields by lets 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 lets look at how this would look if a component is used instead. As can be seen in the example, 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 |

## 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

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.