

Shelter Project Implementation



Activity Description

Application Implementation

Now that we have our data model and some initial data, let's see how our users could interact with it.

Salesforce can manage various business activities including sales and marketing so we will start to create a dedicated "VR" application to separate it from the rest. Salesforce application is composed of tabs. We will start by creating those tabs.

Steps

1

Creating the tabs

Go to **Setup Home**. In the **Quick Find** field, type "**tabs**" and navigate to **User Interface > Tabs**.

Click **New** in the **Custom Object Tabs** section.

Select the **Brief** object and pick any Tab style. Click on **Next**.

Scroll down and click on **Next**.

Click on **Save**.

Repeat the same process for the Configuration object to create a tab for it.

2

Creating the application

In the **Quick Find** field, type "**apps**" and navigate to **Apps > App Manager**.

Click on the **New Lightning App** button on the top right.

Set the following values:

App Name	Shelter Project
Developer Name	Shelter_Project
Image	Upload this image: https://raw.githubusercontent.com/pozil/salesforce-wef-vr/master/workshop-material/app-icon-128x128.png

Click **Next** at the bottom of the screen.

Click **Next** again.

Click **Next** again.

Move the **Brief** and **Configuration** tabs from the Available Items to the Selected Items list, by selecting and clicking on the arrow.


Click **Next**.

Shelter Project Implementation: Page 2

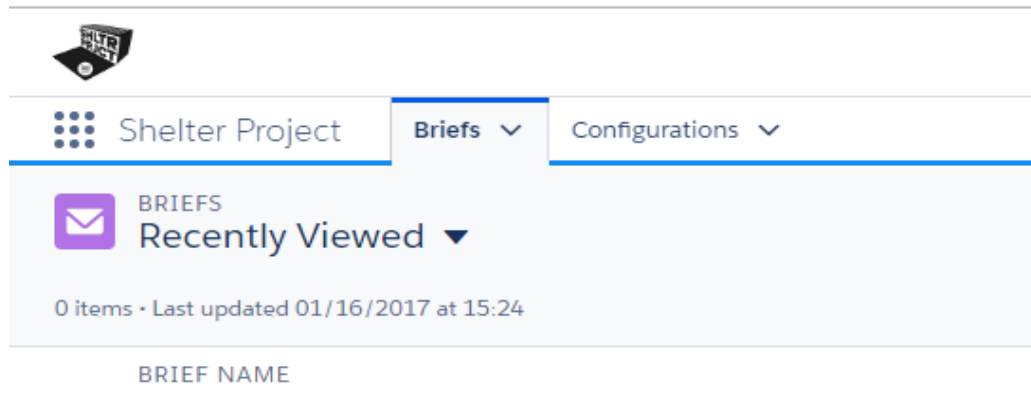


Move the **System Administrator** profile from the Available Profiles to the Selected Profiles list.

Click on **Save** and **Finish**.

Click on the **App Launcher** icon  and select the **Shelter Project** application.
Make sure that the **Briefs** tab is selected.

You should see this:



Notice that the default tab layout can display a list of records and it even has a **New** button to create new records.

Click on the **New** button and you will notice that the default layout is not very convenient to use.

We will not use this default view, we are going to create our own custom view with some code.

3

To test before we get coding, we will create a test brief.

Set the following values:

Brief Name	Test brief
Client	UNHCR
Delivery Date	1/18/2017
Budget	10000
Country	Nepal

Town	Kathmandu
Climate	Wet
Available Space	2
People Sheltered	10

Click on **Save**.



4

Creating a custom Brief list view

In this section, we will replace the default brief list with a custom view that will look like this:

The screenshot shows the Salesforce interface for the 'Shelter Project'. At the top, there is a search bar labeled 'Search Salesforce' and several utility icons. Below the search bar, there are tabs for 'Briefs' and 'Configurations'. The 'Briefs' tab is active, showing a 'BRIEF EDITOR' header. Below the header is a table with the following data:

NAME	CLIENT	LOCATION	CLIMATE	BUDGET	DELIVERY DATE	PEOPLE SHELTERED	ACTION
Primary Brief	UNHCR	Davos (Switzerland)	Cold	\$200,000	Jan 17, 2017	15	
Urgent project	UNICEF	Kabul (Afghanistan)	Warm	\$5,000	Jan 4, 2017	10	

Coding the Console

In order to code our new view, we are going to use the Developer Console.

To open it, click on the **Setup** Cog icon  and select **Developer Console**.

Getting started with component design

Applications are made up of components. We will now create a component for our Brief list view.

In the **Developer Console**, access the following menu **File > New > Lightning Component**.

Name the component BriefListView, **tick** the **Lightning Tab** checkbox then click on **Submit**.

This will open 2 new tabs in the Developer console. You can close the tab named BriefListView, we will only work on **BriefListView.cmp** (this is the component code).

Note: if you close a tab by mistake, you can get back to your component from the **File > Open Lightning Resources** menu

The component's code uses a language called XML that is a superset of HTML (the language in which every web page is made of).



i

XML Reference Sheet

Before moving on, here are a couple of terms we need to understand.

Tags

Tags are building blocks that are used to structure your page. They are enclosed by `<>`.

The generated code contains an opening tag `<aura:component>` and a closing tag `</aura:component>` (notice the `/` character that indicates the closing).

Every opening tag must always be matched with a closing one. The only exception to this rule are self-closing tags such as this one `` (notice the `/` character).

Tags with an opening and closing can incorporate children tags such as:

```
<div>
```

```
<p>Hello</p>
```

```
</div>
```

Attributes

Attributes are used to configure tags. They can only be placed within the opening tags or self-closing tags, never in the closing tags.

In the generated code, the opening tag contains an attribute with a value:

```
implements="force:appHostable".
```

Comments

Comments are invisible to users and help developers organize their code.

Comments can be placed in code with this syntax:

```
<!-- Some comments -->
```



5

Adding the page header and testing your work

Now that we have set the terms & concepts, let's start coding our component. We will start by adding a page header.

Copy/paste the following block of code inside the `<aura:component>` tag (between the opening and closing, on line 2):

```
<!-- HEADER START -->
<div class="slds-page-header" role="banner">
  <div class="slds-grid">
    <div class="slds-col slds-has-flexi-truncate">
      <div class="slds-media slds-no-space slds-grow">
        <div class="slds-media__figure">
          <lightning:icon iconName="custom:custom62" size="medium"/>
        </div>
        <div class="slds-media__body">
          <p class="slds-text-title--caps slds-m-around--x-small">Brief Editor</p>
        </div>
      </div>
    </div>
  </div>
</div>
<!-- HEADER END -->
```

Save the component by using the **Ctrl + s** keyboard shortcut. Remember this shortcut as you will use it often.

Now we want to preview our component and to do that quickly we will create a test application.

Access the following menu **File > New > Lightning Application**.

Name the application TestApp and click on **Submit**.

This will open 2 new tabs in the Developer console. You can close the one named TestApp, we will only work on TestApp.app.

Add the following attribute to the `<aura:application>` tag:

```
extends="force:slds"
```

This will activate the Salesforce style (colours, spacing...) for our page.

Add the following tag inside the `<aura:application>` tag:

```
<c:BriefListView/>
```

This tag represents our Brief list component. What we are doing here is adding our component to the application.

The TestApp application code should now look like this:

```
<aura:application extends="force:slds">
  <c:BriefListView/>
</aura:application>
```

Note: at any time, you can correct the formatting of the code with the following keyboard shortcut combination:

Ctrl + a to select all of the code

Shift + Tab to re-indent the selected code



Ctrl + s to save the code.

To preview your application, click on the blue **Preview** button on the right side of the window.

This will open a new tab in Chrome with the TestApp application. You should see this:



Note: you can preview your work at any time by saving your code in the Developer Console and refreshing this preview tab (**Cmd + r**).

Go back to the Developer Console and select the BriefListView.cmp tab.

6

Adding the page body

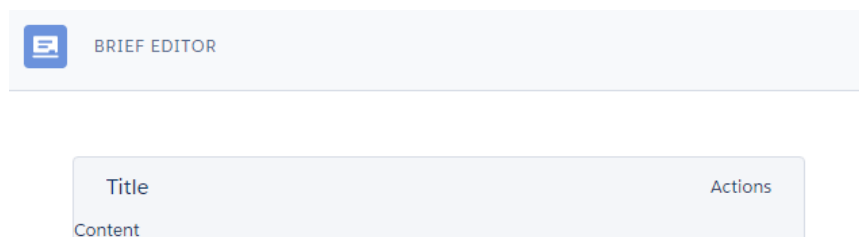
Now that we have the page header, we will add the body of our page.

We will use a component named Card to structure it.

To add the Card component, copy/paste the following code after the page header block.

```
<!-- CARD START -->
<lightning:card class="slds-m-around--xx-large">
  <aura:set attribute="title">
    Title
  </aura:set>
  <aura:set attribute="actions">
    Actions
  </aura:set>
  <!-- CARD CONTENT START -->
  Content
  <!-- CARD CONTENT END -->
</lightning:card>
<!-- CARD END -->
```

Save your code and preview the TestApp application. It should now look like this:



We are now going to set the card's title, actions and content.

For the title, we will have "Briefs" with a counter set to zero for now.

To create it, replace the "Title" text with this code:

```
Briefs <lightning:badge label="0"/>
```

For the actions, we will just have a New button.



To create it, replace the “Actions” text with this code:

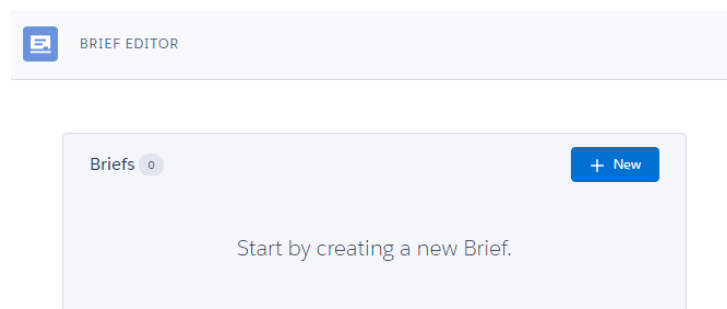
```
<lightning:button variant="brand" label="New" iconName="utility:add"
iconPosition="left"/>
```

For the content section, we will place a message that will guide the user. Since the list is empty, we want to invite them to create a new brief.

Replace the “Content” text with this block of code:

```
<div class="slds-text-heading--medium slds-m-around--xx-large slds-align--absolute-center">Start by creating a new Brief.</div>
```

Save your code and preview the TestApp application. It should now look like this:



7

Adding some data to our page

Our current page is quite static: we are not listing Briefs yet and the New button does nothing. To be able to move forward, we need to make our component dynamic by adding some data to it.

To do so we will use Lightning attributes. These are a specific tag that indicates that we are storing data, such as the custom objects that we created earlier. These are not to be confused with tag attributes.

Declare a list of Briefs custom objects on our BriefListView component by adding the following tag under the top `aura:component` opening tag:

```
<aura:attribute name="briefs" type="Brief_c[]"/>
```

Now that we are storing a list of briefs, we will replace our counter next to the card title with a dynamic expression that will reflect the number of briefs.

Locate the `lightning:badge` tag inside the card title and change the value of its `label` attribute as following:

```
<lightning:badge label="{!v.briefs.length}"/>
```

If you save your work and test the application right now you will notice that the counter still displays zero although we have created a test brief earlier. That is because the brief list Lightning attribute starts empty so we will need to get the data from the server when the page is displayed.

To retrieve data from the server, we need to write a server-side controller.

Navigate to the **File > New > Apex Class** menu.

Name the class `BriefController` click **OK** and save it.



Paste the following code between the brackets:

```
@AuraEnabled
public static List<Brief__c> getAllBriefs() {
    return [SELECT Id, Name, Client__c, Country__c, Town__c, Budget__c,
    Delivery_Date__c, People_Sheltered__c, Climate__c FROM Brief__c ORDER BY Name ASC];
}
```

Save, then close the BriefController.apxc tab and select the BriefListView.cmp tab.

We will start by declaring a reference to the server code we just wrote by replacing the opening `aura:component` tag (first line) by this code:

```
<aura:component controller="BriefController" implements="force:appHostable">
```

Then, we will add the call to our server that retrieve our briefs.

Click on the **Controller** button on the right side of the window.

This will open a new tab.

Replace all of its content with the following:

```
((
    doInit : function(component, event, helper) {
        var action = component.get("c.getAllBriefs");
        action.setCallback(this, function(response){
            if (component.isValid() && response.getState() === "SUCCESS")
                component.set("v.briefs", response.getReturnValue());
            else
                console.log(response);
        });
        $A.enqueueAction(action);
    },
))
```

Now that we have the code responsible for loading our data, we just need to call it from our component.

Save, then close the BriefListViewController.js tab and got back to the BriefListView.cmp tab.

We will load our data when the page loads (initializes) by adding this line under the briefs Lightning `aura:attribute`, near the top of the file:

```
<aura:handler name="init" value="{!this}" action="{!c.doInit}" />
```

This handler tag will call the `doInit` function we have declared in BriefListViewController.js when the page is displayed. The `doInit` function will then call the server to retrieve all of the available briefs (`c.getAllBriefs`) and store them in the `briefs` Lightning attribute. Finally, when the component is displayed, it will calculate the length of the brief list and display it in the counter.

Save your work and test the application again.

You should notice that the brief counter indicates "1" instead of "0".



8

Displaying the list of briefs

We now have access to the list of available briefs so let's display it in a table.

We will keep the current "Start by creating a new Brief" invite but we will only display it when the brief list empty (that is not the case now since we have our test brief).

To display or hide the text dynamically, based on the brief list length, we are going wrap some code around it.

Place the following line before the `div` tag that contains the current welcome text (at the start of the card content):

```
<aura:if isTrue="{!v.briefs.length == 0}">
```

Add the following code after the `div` tag:

```
<aura:set attribute="else">
  Brief table
</aura:set>
</aura:if>
```

Save your work and test the application.

You should now see "Brief table" displayed (poorly formatted) instead of the invite text.


Replace the "Brief table" text with the following block of code:

```
<!-- TABLE START -->
<table class="slds-table slds-table--bordered slds-table--cell-buffer slds-table--striped">
  <!-- TABLE HEAD START -->
  <thead>
    <tr class="slds-text-title--caps">
      <th>
        <div class="slds-truncate" title="Name">Name</div>
      </th>
      <th>
        <div class="slds-truncate" title="Client">Client</div>
      </th>
    </tr>
  </thead>
  <!-- TABLE HEAD END -->
  <!-- TABLE BODY START -->
  <tbody>
    <tr>
      <td data-label="Name">
        <div class="slds-truncate">
          <h3 class="slds-text-heading--small">Name Value</h3>
        </div>
      </td>
      <td data-label="Client">
        <div class="slds-truncate">
          Client value
        </div>
      </td>
    </tr>
  </tbody>
  <!-- TABLE BODY END -->
</table>
<!-- TABLE END -->
```

Save your work and test the application.



It should now look like this:

 BRIEF EDITOR

Briefs 1		+ New
NAME	CLIENT	
Name value	Client value	

We now have a table with two columns (Name and Client), a header row, a second row.

Look closely at the code structure to find a pattern and try to add the following columns on your own:

- Location
- Climate
- Budget
- Delivery Date
- People Sheltered

Tip: when you add a column in the table header, you must also add the matching column in the table second row.

Save your work and test the application from time to time.

Your table should look like this once you are done:

NAME	CLIENT	LOCATION	CLIMATE	BUDGET	DELIVERY DATE	PEOPLE SHELTERED
Name value	Client value	Location value	Climate value	Budget value	Delivery Date value	People Sheltered value

The table looks good but the data inside it is not dynamic yet.

Firstly, there will always be a single row no matter how many briefs we have. We need to make the number of rows match the number of available briefs dynamically.

To change that, we are going to add a tag that allows us to repeat a part of the page: the table row.

Locate the `<!-- TABLE BODY START -->` comment and add the following block of code after the opening `tbody` tag:

```
<aura:iteration items="{!v.briefs}" var="brief">
```

Now, we need to add the closing `aura:iteration` tag. Scroll down and locate the `<!-- TABLE BODY END -->` comment. Locate the `tbody` closing tag, then add the following block of code above this closing `tbody` tag:

```
</aura:iteration>
```



At this point you will not see a difference if you preview the application: there will still be a single row but that is because we only have one brief record (our test brief).

The `aura:iteration` tag will repeat the content, that lies between its opening and closing tag, as many times as there are items. Each time it repeats, it will copy a value of items to the variable specified in the var attribute.

That means that in our case we can use the brief variable object to display the data of a brief.

We can now replace the "... value" text of each column with the brief values.

To display the value of a variable (myVariable for example) we need to write an expression of this form: `{!myVariable}`

In our case, we are using an object with fields, so we will need to use the following expression:

```
{!brief.Name}
```

In the above example brief is the object and Name is the field that we will display.

Replace all column placeholder values with dynamic expressions.

As a reminder, here are the names of the different brief object fields we will use:

- Name
- Client__c
- Town__c
- Climate__c
- Budget__c
- Delivery_Date__c
- People_Sheltered__c

Once you are done, save your work and test the application.

The table should now look like this:

NAME	CLIENT	LOCATION	CLIMATE	BUDGET	DELIVERY DATE	PEOPLE SHELTERED
Test brief	UNHCR	Kathmandu	Wet	10000	2017-01-18	10

9

The table is starting to look good but we can still improve it.

We will start by detailing the location, by adding the country between parenthesis after town.

Locate the town column and add this after the town expression:

```
{{!brief.Country__c}}
```

Next, we will take care of the format of the budget column.

Replace its value expression with the following currency formatting tag:

```
<ui:outputCurrency value="{!brief.Budget__c}" format="$#,###"/>
```



We will apply the same kind of technique for the Delivery Date.

Replace its value expression with the following currency formatting tag:

```
<ui:outputDate value="{!brief.Delivery_Date__c}"/>
```

Save your work and test the application.

The table should now look like this:

NAME	CLIENT	LOCATION	CLIMATE	BUDGET	DELIVERY DATE	PEOPLE SHELTERED
Test brief	UNHCR	Kathmandu (Nepal)	Wet	\$10,000	Jan 18, 2017	10

Congratulations, you have implemented a dynamic brief view!

10

Deploying the complete application

What we have coded for the past hour is just a small part of the application that we will use for today's work. We will now deploy the complete application, but before we can do that we will need to remove what we did first.

11

Removing our application

In the Developer Console, select the **TestApp.app** tab. If you do not see it, you can open it with the **File > Open Lightning Resources** menu.

Navigate to the **File > Delete** menu and confirm the deletion.

Select the **BriefListView.cmp** tab use the same menu to **delete** it as well.

Open the **BriefController controller** by using the **File > Open** menu and **delete** it.

Close the Developer Console.

Open the **Shelter Project application** if it is not open and navigate to the **Briefs** tab.

Click on the arrow facing down, on the right side of the row of your Test brief.

Select **Delete** and confirm the deletion.

Go into **Setup**.

Type "tabs" in the Quick Find field and navigate to **User Interface > Tab**.

Click on the **Del link** next to your two custom tabs and remove them both.



12


Deploying the complete version


Now that we have removed our version of the application we can import the complete application.

To do so, click on [this link](#).


Click on Allow, then click on Deploy in the top right corner of the window.

Wait a couple of seconds or minutes until you see the text "Deployment Complete" like this:


GitHub Salesforce Deploy Tool
Deploy directly from GitHub to Salesforce [andyinthecloud](#)
Deploy


From GitHub Repository

Name: pozil/salesforce-wef-vr-completed
Description: Completed Salesforce WEF VR Workshop
URL: <https://github.com/pozil/salesforce-wef-vr-completed>


To Salesforce Org

Organization Name: WEF VR 01
User Name: admin@wef-vr-01.com

Deployment Started
Status: Queued
Status: InProgress
Status: InProgress
Status: InProgress
Status: InProgress
Status: Completed
Deployment Complete
[src](#)
[src/package.xml](#)
[src/aura](#)
[src/aura/BriefEditionModal](#)

You can now close the Chrome tab.

Go back to Setup.

Type "tabs" in the Quick Find field and navigate to User Interface > Tab.

Click New in the the Lightning component tabs section.

Set the following values:

Lightning Component	c:BriefListView
Tab Label	Briefs
Tab Name	Briefs
Tab Style	<i>Pick any style</i>

Click Next then click Save.

Repeat the same operation to create a second Lightning component tab with these



values:

Lightning Component	c:ConfigurationEditor
Tab Label	Configurations
Tab Name	Configurations
Tab Style	<i>Pick any style</i>

Type "apps" in the Quick Find field and navigate to Apps > Apps Manager.

Locate the Shelter Project application row, click on the arrow on the right of the row and select Edit.

Click on the "Select Items" tab.

Move the Briefs and Configurations tabs from the Available Items to the Selected Items list.

Click Save then click Done.

Refresh the browser page and navigate to the Shelter Project application.

Congratulations, you are now using the complete Shelter Project application and you're ready to begin creating a configuration in response to the brief.