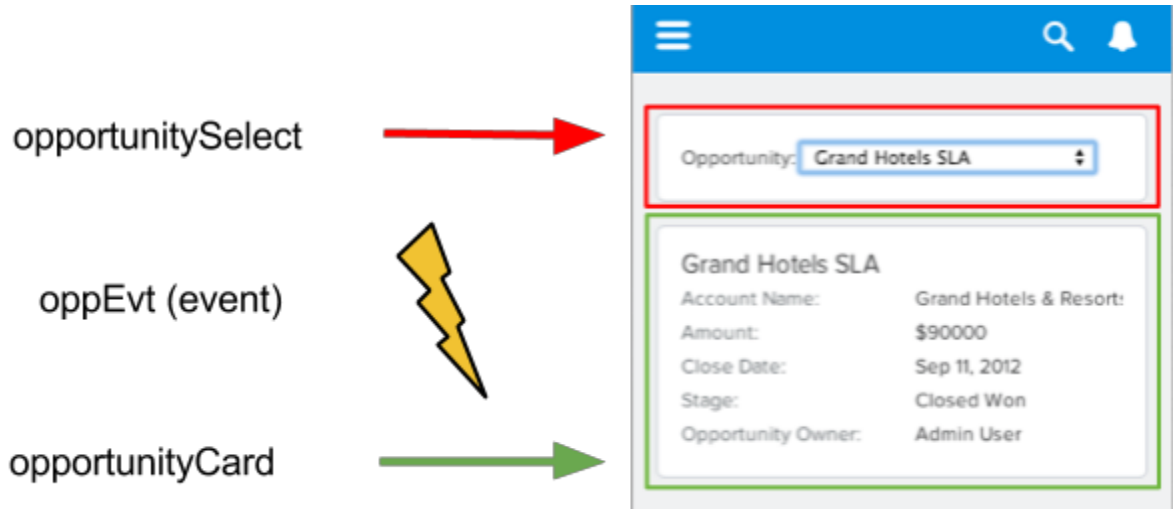
Lightning Components Mini Workshop

In this mini workshop you build an app that listens for changes to the Opportunity standard object.

The app consists of two components and an event. The top portion (`opportunitySelect`) is used for selecting an opportunity. The bottom component (`opportunityCard`) is a card that displays information. These components are loosely coupled; the event is what connects them.



- `opportunitySelect` component (top) - Talks to Salesforce and brings in a list of opportunities. If a user selects or changes an opportunity, it fires the selectOpportunity event with the Opportunity Id.
- `oppEvt` event - This is an event that `opportunitySelect` controller fires, and that `opportunityCard` listens to.
- `opportunityCard` component (bottom) - Listens to the oppEvt event and receives Opportunity Id. Sends Id to Salesforce and gets Opportunity Details. Displays the details in a card.

Tutorial 1: Initial Set Up

Before you begin you to need to get a new Developer Edition organization, create a namespace, and enable Lightning Components for your org.

Step 1: Sign Up For Developer Edition

In order to use Lightning Components, you must have a Salesforce organization that is running the Winter '15 version. The easiest way to make sure is to sign up for a new Developer Edition org.

1. Go to https://bit.ly/lightningcomponents

2. Fill out the form, using an email address you can easily check right now. The username is in the form of an email address, but it doesn't have to be the same as your email address. *your.name@DF14.com* is a perfectly good username, even if it isn't your actual email address.

Step 2: Assign a Namespace

Every Lightning component is prefixed by a namespace, so the first thing you need to do is register for a unique namespace of your own.

1. Click **Setup > Create > Packages**, and then click **Edit.**
2. Agree by clicking **Continue.**
3. Type a namespace and then click **Check Availability** to see if it's unique.
4. Click **Review My Selections** and then **Save**.

Step 3: Enable Lightning Components

1. From **Setup**, click **Develop | Lightning Components.**
2. Select the checkboxes for **Enable Lightning Components** and **Enable Debug Mode** and then **Save**.

Tutorial 2: Build the OpportunityPanel App

In this section you will:

- Create a new Apex Controller Class to return a list of Opportunities
- Create an `opportunitySelect` Lightning component
- Display Salesforce Data in the component

Step 1: Create an Apex Controller

First you need to create an Apex (server-side) controller to return a list of opportunities.

1. Click Select **File > New > Apex Class**
2. For the class name, enter *OpportunityController* and click **OK**.
3. Add the following code to the class.

```
public class OpportunityController {

    @AuraEnabled
    public static List<Opportunity> getOpportunities() {
        List<Opportunity> opportunities =
           [SELECT Id, Name, CloseDate FROM Opportunity];
        return opportunities;
    }
}
```

Tell Me More….

- The `@AuraEnabled` annotation makes the `getOpportunies` method accessible to Lightning components.

Step 2: Read the List of Opportunities from the Server

The `opportunitySelect` component will be used to read the list of opportunities from the server.

1. In the Developer Console, select **File | New | Lightning Component**.
2. For the name, enter `opportunitySelect` and then click **Submit**.
3. Paste in the following code, making sure to replace <namespace> with your own.

```
<aura:component controller="<namespace>.OpportunityController">
    <aura:attribute name="opportunities" type="Opportunity[]"/>
    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
    <div class="controlLayout">
        <ui:outputText value="Opportunity:" class="truncate"/>
        <select aura:id="opportunitySelect">
                  <aura:iteration var="opportunity"
items="{!v.opportunities}">
                <option
value="{!opportunity.Id}">{!opportunity.Name}</option>
            </aura:iteration>
```

```
            </select>
        </div>
    </aura:component>
```

Tell Me More….

Take a look at what each part of the code is doing:

- `<aura:component controller="<namespace>.OpportunityController">` - Connects the component to the `OpportunityController` Apex class.
- `<aura:handler name="init" value="{!this}" action="{!c.doInit}" />` - Calls the `doInit` JS Controller function on component initialization. The `{!c.doInit}` expression syntax calls `doInit` function in the client-side controller
- `<aura:attribute name="opportunities" type="Opportunity[]"/>` - Creates an opportunities attribute to store the list of opportunities.
- `<aura:iteration var="opportunity" items="{!v.opportunities}">` - Loops through the list of opportunities and adds them to an HTML list.

Step 3: Create the `opportunitySelect` Component

Create the `opportunitySelect` component to read the list of opportunities from the server.

1. Select **File | New | Lightning Component.**
2. For the name, enter `opportunitySelect` and click **Submit**.
3. Paste in the following code, making sure to change the namespace to your own.

```
<aura:component controller="<namespace>.OpportunityController">
    <aura:attribute name="opportunities" type="Opportunity[]"/>
    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
    <div class="controlLayout">
        <ui:outputText value="Opportunity:" class="truncate"/>
        <select aura:id="opportunitySelect">
                  <aura:iteration var="opportunity"
items="{!v.opportunities}">
                <option value="{!opportunity.Id}">{!opportunity.Name}</option>
            </aura:iteration>
        </select>
    </div>
</aura:component>
```

Tell Me More….

Take a look at some of the code and what it's doing.

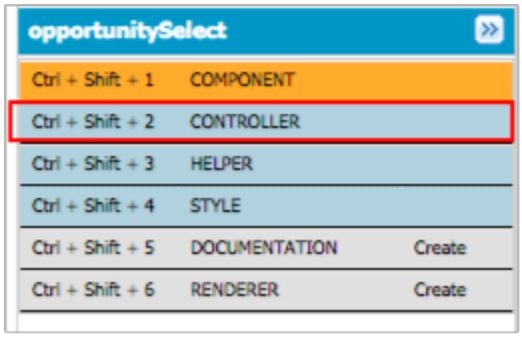
- `<aura:component controller="<namespace>.OpportunityController">` - This connects the component to the `OpportunityController` Apex class.

- `<aura:handler name="init" value="{!this}" action="{!c.doInit}" />` - This Calls the doInit JS Controller function on component initialization. The `{!c.doInit}` expression syntax calls doInit function in the client-side controller
- `<aura:attribute name="opportunities" type="Opportunity[]"/>` - This creates an opportunities attribute to store the list of opportunities.
- `<aura:iteration var="opportunity" items="{!v.opportunities}">` - This loops through the list of opportunities and adds them to an HTML list.

Step 4: Add a JavaScript Controller

The JavaScript controller contains multiuple JS functions also known as Actions.

1. Click CONTROLLER in the Sidebar to create a JS controller.



2. Paste in the following code:

```
({
    doInit: function(component, evt, helper) {
       var action = component.get("c.getOpportunities");
        action.setCallback(this, function(a) {
            var opportunities = a.getReturnValue();
            component.set("v.opportunities", opportunities);
        });
        $A.enqueueAction(action);
    }
})
```

Tell Me More….

Take a look at what the code is doing:

- `doInit: function` - Called with the component is initialized.
- `var action = component.get("c.getOpportunities")` – Creates an action object to call getOpportunities on the Apex controller.
- `action.setCallback(this, function(a)` – A callback function to get a list of opportunities.
- `var opportunities = a.getReturnValue()` - Stores the list of opportunities from the server.
- `component.set` – Updates the view.

- `$A.enqueueAction(action)` - Asks Lightning framework to make an AJAX call when it can. The code in action.setCallback is run after the response comes back from the `$A.enqueueAction`. Note that `$A` is shorthand for “Aura” - a global Aura Object.

Step 5: Test the Component

Components can run in three different places:

- Inside a “Standalone Lightning App” - Like the helloworld app you created earlier.
- Inside the Salesforce1 mobile app - You’ll do this in a moment.
- As a Component Extension for the Salesforce1 mobile app - Here you simply replace an existing Salesforce1 mobile app’s own component with the one you create.

Start by creating a standalone Lightning app to test the component.

Step 6: Create a Lightning App Bundle

A Lightning Component contains multiple resource files together known as “bundles”.

A bundle typically contains:

- Component - *name*.cmp or *name*.app
- Controller - *name*Controller.js
- Helper - *name*Helper.js
- Style - *name*.css

In this step you create an Application Bundle.

1. In the Developer Console, select **File > New Lightning Application.**
2. For the application name, type `opportunityStandAlone`.
3. Click **Submit.**
4. Copy and paste the following code, making sure to update `<namespace>` to your own.

```
<aura:application>
      <<namespace>:opportunitySelect/>
</aura:application>
```

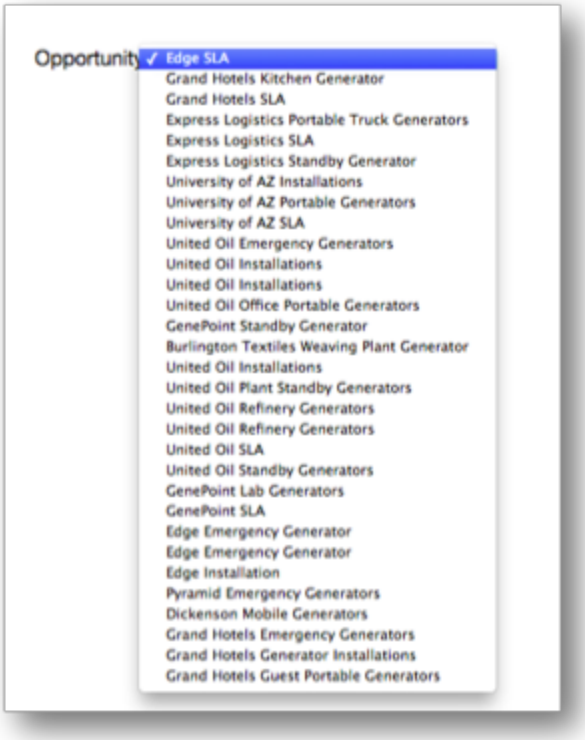
5. Click STYLE and update the App’s background from #000000 to white.

6. **Save**.

Step 7: Preview the App

There's more than one way to preview your app. If you've used Visualforce before, you're probably familiar with modifying the URL in your browser to preview pages by using `https://<instance>.salesforce.com/apex/<page_name>`
You can do the same thing with Lightning apps by entering the following in your browser: `https://<instance>.salesforce.com/<namespace>/<appname>.app`

1. Try it out by opening a new browser tab and appending your namespace and app name at the end.
   `https://<instance>.salesforce.com/<namespace>/<appname>.app`
2. You can also click the **Preview** button in the Components Sidebar and you should see a new window open with your app.
3. You should see a list of opportunities. Expand the list, it should look like the following image.



Tell Me More….

- The Salesforce1 mobile app is also a standalone Lightning application! You can access it by using `https://<salesforce_instance>/one/one.app`. As you might have figured, the namespace is `one`, and the application is called `one.app`.
- When you enter `<instance>.salesforce.com/<namespace>/<app_name>` it's actually redirected to `<instance>.lightning.force.com/<namespace>/<app_name>`.

Tutorial 3: Create Events

Events allow a component to send and receive messages. There are two types of events in Lightning Component framework.

- APPLICATION level event - This can be sent and received by any component in the application (i.e. global).
- COMPONENT level event - This can only be sent and received by component or component hierarchy

For this tutorial you will use an APPLICATION level event.

You will also need to create a JavaScript Helper. Helpers contains JS functions that can be called by any JS code in the bundle. In this tutorial it is used to fire an event from `opportunitySelect.cmp`.

Step 1: Create the `oppEvt` Event

For starters, create the application-level event.

1. In the Developer Console, select **File > New Lightning Event.**
2. For the event name, type `oppEvt` and click **Submit.**
3. Copy and paste the following code.

```
<aura:event type="APPLICATION">
      <aura:attribute name="id" type="String"/>
</aura:event>
```

Step 2: Add a JavaScript Helper for `opportunitySelect`

1. In the sidebar, select `opportunitySelect.cmp` (the component file).
2. Click HELPER on the sidebar.
3. Copy Paste the code below, recalling that you must change the namespace to that of your own.

```
({
    fireSelectOpportunity : function(id) {
    // Called by Controller with Opportunity Id
        var oppEvt = $A.get("e.<namespace>:oppEvt");
        // Grab oppEvt event object
        oppEvt.setParams({id: id}); //Set opportunity Id
        oppEvt.fire(); // Fire the event
  }
})
```

Tell Me More….

Take a look at what the code is doing.

- `oppEvt` function is called when user selects an opportunity item (or on initial load). The rest of the code is commented out for now. We will uncomment it soon.
- `$A.get()` is used to get any artifacts within the Lightning Components system.
- `$A.get("e.<namespace>:oppEvt")` returns the `oppEvt` event (you will create it soon).

Step 3: Update opportunitySelect.cmp

1. Open `opportunitySelect.cmp.`
2. Update the file with the following code:

```
<aura:component controller="<namespace>.OpportunityController">
    <aura:attribute name="opportunities" type="Opportunity[]"/>
    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
    <div class="controlLayout">
        <ui:outputText value="Opportunity:" class="truncate"/>
        <select aura:id="opportunitySelect"
onchange="{!c.onSelectOpportunity}">
                  <aura:iteration var="opportunity"
items="{!v.opportunities}">
               <option value="{!opportunity.Id}">{!opportunity.Name}</option>
            </aura:iteration>
        </select>
    </div>
</aura:component>
```

Tell Me More….

- `<select aura:id="opportunitySelect" onchange="{!c.onSelectOpportunity}">` - Calls JS controller when user changes the opportunity in the list (hint: in turn fires an event)

Step 4: Update the Controller to Use the Helper

1. Open `opportunitySelectController.js` (part of `opportunityController.cmp`)
2. Replace all its code with the following code and then **Save**.

```
({
    doInit: function(component, evt, helper) {
       var action = component.get("c.getOpportunities");
        action.setCallback(this, function(a) {
            var opportunities = a.getReturnValue();
            component.set("v.opportunities", opportunities);
            helper.fireSelectOpportunity(opportunities[0].Id);
        });
        $A.enqueueAction(action);
    },
    onSelectOpportunity : function(component, event, helper) {
```

```
        // Used by OpportunityCard (you will create it next)
        var select = component.find("opportunitySelect").getElement();
        var value = select.value;
        helper.fireSelectOpportunity(value);
    }
})
```

Tutorial 4: Build the `OpportunityCard` Component

Step 1: Create the Component

1. In the Developer Console, select **File > New > Lightning Component.**
2. For the name, type `opportunityCard` and click **Submit.**
3. Copy and paste the following code, making sure to use your own namespace.

```
<aura:component controller="<namespace>.OpportunityController">
    <aura:attribute name="record" type="Opportunity" default="{ 'sobjectType':
'Opportunity' }"/>
    <aura:handler event="<namespace>:oppEvt"
action="{!c.handleSelectOpportunity}" />
    <div class="listRecord recordLayout">
        <div class="itemTitle">
            <ui:outputText class="itemTitle" value="{!v.record.Name}"/>
        </div>
        <div class="recordItem">
            <ui:outputText value="Account Name:" class="recordCell label
truncate"/>
            <ui:outputText value="{!v.record.Account.Name}" class="recordCell
value truncate"/>
        </div>
        <div class="recordItem">
            <ui:outputText value="Amount:" class="recordCell label truncate"/>
            <ui:outputText value="{!'$' + v.record.Amount}" class="recordCell
value truncate"/>
        </div>
        <div class="recordItem">
            <ui:outputText value="Close Date:" class="recordCell label
truncate"/>
            <ui:outputDate value="{!v.record.CloseDate}" class="recordCell
value truncate"/>
        </div>
        <div class="recordItem">
            <ui:outputText value="Stage:" class="recordCell label truncate"/>
```

```
            <ui:outputText value="{!v.record.StageName}" class="recordCell
value truncate"/>
        </div>
        <div class="recordItem">
            <ui:outputText value="Opportunity Owner:" class="recordCell label
truncate"/>
            <ui:outputText value="{!v.record.Owner.Name}" class="recordCell
value truncate"/>
        </div>
    </div>
</aura:component>
```

Step 2: Create the Component Controller

Select `opportunityCard.cmp`.

1. Click CONTROLLER in the sidebar menu
2. Copy and paste the following code:

```
({
      handleSelectOpportunity: function(component, evt, helper) {
            var id = evt.getParam("id");
            var action = component.get("c.getOpportunity");
            action.setParams({id: id});
            action.setCallback(component, function(a) {
                  var record = a.getReturnValue();
                  component.set("v.record", record);
            });
            $A.enqueueAction(action);
      }
})
```

Step 3: Update the Server-side Controller

Next you need to update the `OpportunityController.apxc` to return details for a given Id.

1. Open `OpportunityController.apxc`
2. Remove all the code.
3. Paste in the following code:

```
public class OpportunityController {

    @AuraEnabled
    public static List<Opportunity> getOpportunities() {
        List<Opportunity> opportunities = [SELECT Id, Name, CloseDate FROM
Opportunity];
        return opportunities;
```

```
    }

    @AuraEnabled
    public static Opportunity getOpportunity(Id id) {
        Opportunity opportunity = null;
        if (id != null) {
            opportunity = [
                SELECT Id, Account.Name, Name, CloseDate, Owner.Name, Amount,
Description, StageName
                FROM Opportunity
                WHERE Id = :id
            ];
        }
        return opportunity;
    }
}
```

Tell Me More….

Take a look at what the code is doing.

- `<aura:component controller="<namespace>.OpportunityController">` - Binds this component to the `OpportunityController` Apex (server-side) Controller
- `<aura:attribute name="record" type="Opportunity" default="{ 'sobjectType': 'Opportunity' }"/>` - Creates a record attribute whose type is Opportunity and also specifies that actual data in the server's JSON response will be in 'sobjectType' key.
- `<aura:handler event="<namespace>:oppEvt" action="{!c.handleSelectOpportunity}" />` - Declares that this component handles the `oppEvt` event and calls `handleSelectOpportunity` JS controller function.

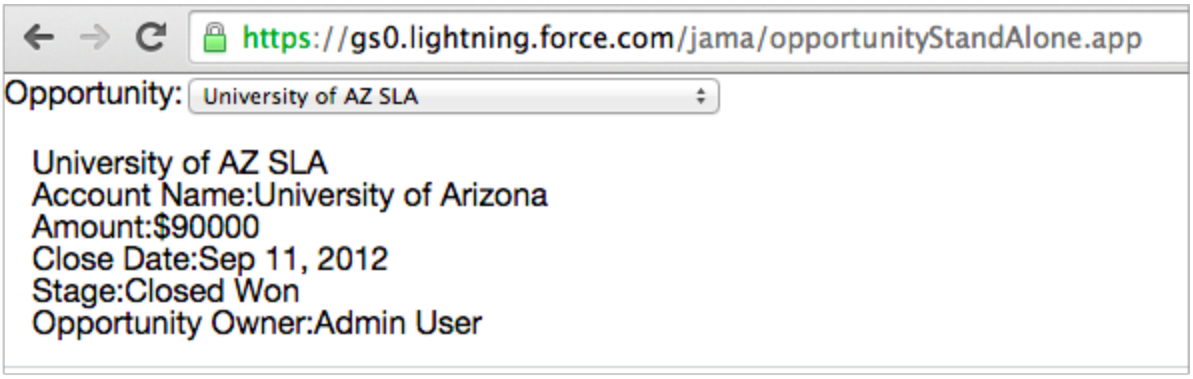
Step 4: Add the Card to the App

1. Select `opportunityStandAlone.app` that you created earlier.
2. Remove all of the code and then paste in the following. Make sure to update the namespace and **Save** the file.

```
<aura:application>

      <<namespace>:opportunitySelect/>

      <<namespace>:opportunityCard/>

</aura:application>
```

Step 5: Test the App

1. In the sidebar, click **Update Preview**.
2. Select or change an opportunity in the `opportunitySelect` component and you should see the details, as in the following image.



Tutorial 5: Create the Salesforce1 Component

Step 1: Create a New Lightning Component

1. In the Developer Console, select **File | New**.
2. For the name, enter `opportunityPanel` and click **Submit**.
3. Select `opportunityPanel.cmp`
4. Copy and paste the following code. Make sure to change the namespace and **Save**.

```
<aura:component implements="force:appHostable">
      <div class="container">
      <div aura:id="content" class="center content">
            <div>
                <<namespace>:opportunitySelect/>
                  <div aura:id="cards">
                  <<namespace>:opportunityCard/>
                  </div>
            </div>
      </div>
      </div>
</aura:component>
```

Step 2: Create Tabs

1. Switch from the Developer console to the browser.
2. Click **Setup > Create > Tabs**.
3. In the Lightning Components Tabs section, click **New**.
4. Select `<namespace>:opportunityPanel` from the selector
5. Set the Tab Label to Opportunity Panel.
6. Set the Tab Name to Opportunity_Panel.
7. Choose a Tab Style (anything will do) and click **Next**.
8. **Save**.

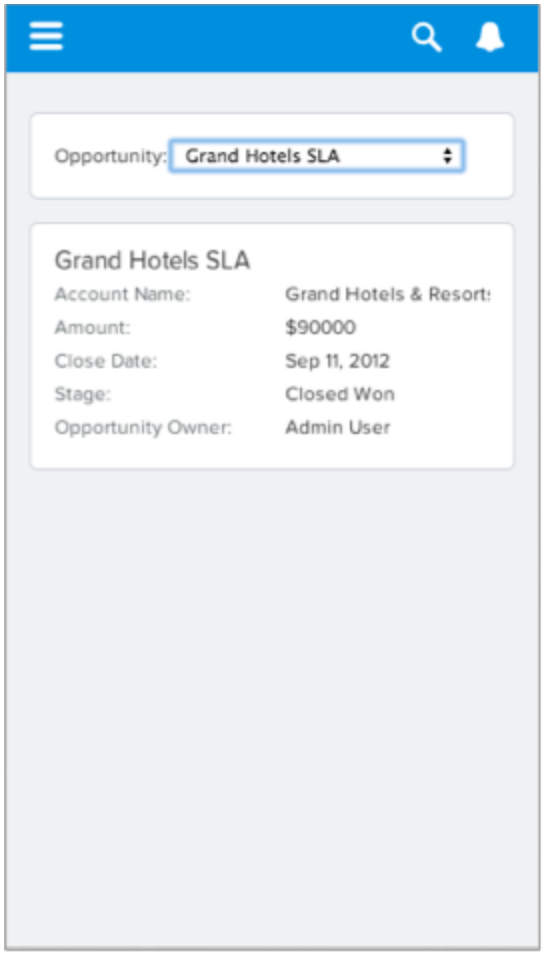
Step 3: Set up Mobile Navigation

1. In Setup, type `mobile n` in the search box.
2. Select **Mobile Administration -> Mobile Navigation.**
3. Select **Opportunity Panel** in the Available list and **Add** it to the Selected list.
4. Select Opportunity Panel in the Selected list click the **Up** button to move it below People.
5. **Save**.

Step 4: Try it Out

In a browser window, navigate to the S1 Mobile App app by appending `one/one.app` to your Salesforce instance.

1. Expand the navigator by clicking the icon.
2. Select Opportunity Panel from the list.



Congratulations! That concludes the mini workshop. Make sure to go to the Developer Library and pick up a copy of the Lightning Components Development Guide and Cheat Sheets.

Appendix - Adding Style

So far the components you created are functional, but plain. Each component needs to use its own CSS. The .THIS prefix must be used with all styles, which will map to a generated namespace.

Step 1: Add a CSS to the Card

1. Open opportunityCard.cmp.
2. In the Sidebar, click **STYLE**.
3. Add the following code.

```
.THIS.recordLayout {
    list-style: none;
    padding: 14px;
    border-bottom:1px solid #cfd4d9
}

.THIS.listRecord {
    background-color: #ffffff;
    margin: 14px;
    border-radius: 5px;
    border: 1px solid #cfd4d9;
    position: relative;
}

.THIS .uiInputSelect {
    margin: 0px 4px;
}

.THIS .uiOutputText {
    line-height: 1.1em;
}

.THIS.listRecord button {
    display: inline-block;
    outline: 0px none;
    border: 0px none;
    background: transparent;
    position: absolute;
    min-width: 16px;
    min-height: 16px;
    max-width: 32px;
    max-height: 32px;
    right: 0px;
```

```
    top: 0px;
}
.THIS.listRecord button.trash {
}

.THIS.listRecord button.lock {
    right: 32px;
}

.THIS.listRecord button.unlocked {

}

.THIS.listRecord button.locked {

}

.THIS .pin .label {
    width: auto;
}

.THIS .itemTitle {
    font-size: 15px;
    padding-bottom: 3px;
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}

.THIS .recordCell {
    padding: 1px 14px 1px 0;
    line-height: 16px;
    max-height: 16px;
    display: inline-block;
}

.THIS .recordItem {
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
.THIS .recordItem .label {
    padding-right: 14px;
    width: 125px;
    color: #696e71;
}

.THIS .value {
```

```
}

.THIS .truncate {
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
```

Step 2: Add a CSS to opportunitySelect.cmp

1. Open opportunitySelect.cmp.
2. In the Sidebar, click **STYLE**.
3. Add the following code.

```
.THIS.controlLayout {
    padding: 14px;
    background-color: #ffffff;
    margin: 14px;
    border-radius: 5px;
    border: 1px solid #cfd4d9;
    white-space: nowrap;
}
.THIS.container {
    width: 100%;
    padding: 10px 0px;
    margin: 0px;
}
```

Step 3: Add a CSS to the App

1. Open opportunityStandAlone.app.
2. In the Sidebar, click **STYLE**.
3. Add the following code.

```
.THIS .center {
    margin: 0px auto;
}

.THIS .content {
    max-width: 768px;
    font-size: 12px;
    color: #3c3d3e;
}
```

Step 4: Test the App

1. In the opportunityStandAlone.app sidebar, click **Update Preview**.
2. Select or change an opportunity in the opportunitySelect component and you should see the details, as in the following image.