



HANDS-ON ACTIVITIES

MOBILE SOLUTIONS ARCHITECTURE

FEBRUARY, 2018



1. COMPREHENSIVE USE CASE

Universal Waters (UW) is a utility company that provides water distribution and last mile support to both commercial and residential customers. Because they are a utility, they are required to provide an excellent level of service or they run the risk of losing their monopoly in many of their covered regions: 4 northeastern states. Up until recently, they provided support via phone, email, and a legacy “brochureware” website, but they were not able to scale their operations.

Recently they embarked on a Service Cloud implementation to increase the level of support they provide to their customers. Their first implementation phase was geared toward ensuring their internal reps could manage the inbound emails and calls. They used the service console to provide a great desktop experience for their agents. They are now in the process of designing the second phase.

DETAILED REQUIREMENTS

SELF-SERVICE

- Customers will be able to request a service call if something is wrong.
- There are a number of different reasons that a service request can be created.
- Customers can upload a picture associated with the request.
- The customer can share their location with the request.
- A customer can receive updates on their service request.
- App Distribution:
 - Users should be able to access the application on the desktop as well as on a mobile device.
 - They insist that they reach as close to 100% of the smartphone market as possible.
 - They also want to ensure that users will have a solid experience on a tablet device.
 - While smartwatches are not ubiquitous, they are interested in supporting these emerging platforms.
 - They want to distribute the apps on at least the top two (2) major app stores.



SALES TEAM

- There's a small group of Sales Reps who offer water-related products and services to their customers.
- These users should be able to track and update the opportunities on the road without having to go back to the office after every meeting or presentation.
- Their Sales Manager has also asked for instant access to reports to track the progress of her team in order to be able to identify reps who might need her support to close a deal or to push others who have not been actively pursuing new deals.
- The Sales team members will all be given iPads.
- They will also need to see billing history as well as be able to create and submit new contracts on the road. There is an existing multi-step Visualforce "wizard" ending in a synchronous web service call to the ERP system on contract submission.
- The Sales Reps sometimes have to travel to remote areas where there is little to no cellular coverage, but they will still need access to data and content for discussions with the customer.

SERVICE AGENTS

- Service agents handle cases across many channels: email, phone, etc.
- There are ~100 service agents located in 2 service centers.
- The UW agents will spend most of their time at their desk in the office supporting service requests.
- Agents should have the ability to support service requests from wherever they are.
- UW is evaluating mobile providers and is looking for the ability to offer BYOD for their employees, while providing a container to protect company intellectual property.
- The mobile security team insists that they can wipe all corporate data if an employee is terminated or a phone is lost.
- If appropriate, Agent Dispatchers will assign service requests to Field Service Agents depending on proximity to the account and availability.

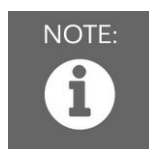


SERVICE PROVIDERS

- Service Providers are independent contractors that respond to the service requests they are given.
- They often go to the commercial or residential customer and diagnose and fix the problems related to the request.
- Service Providers (contractors) will need to be able to respond to service requests.
- Service Providers are not employees and should only see service requests that are assigned to them.
- Service Providers will need to be able to take a picture of a meter, annotate it, and attach it to the case to prove that the service work has been completed.
- Service Providers will need to “sign” a case with TouchID before submitting.
- Service Providers will all be given iPads.

OTHER CONSIDERATIONS

- UW is looking for a rollout strategy for each of its deployment audiences. UW has expressed interest in evaluating an MDM platform.
- UW is looking for a recommendation for how to contract with development teams; i.e., what skill sets are required.
- UW is looking to minimize maintenance where possible, but is willing to concede to meet the core requirements.
- Part of this effort will be to refresh the public-facing website and to provide a pleasant UX for all users on all platforms.
- UW wants to enable SSO to facilitate good user adoption. They are planning to manage this through a SAML-based authentication. They utilize AD for internal users and LDAP for customers. Contractors are considered internal.



NOTE:

Please note that the build materials for this domain should be completed in order, as each one builds upon the previous one.



2. MOBILE ARCHITECTURAL DESIGN

Describe the design considerations, trade-offs, and risks for mobile solutions and recommend the appropriate mobile platform: HTML5, Native (iOS/android/Windows Mobile), Hybrid solutions, or Salesforce1 Mobile App.

PREREQUISITE SETUP STEPS

N/A

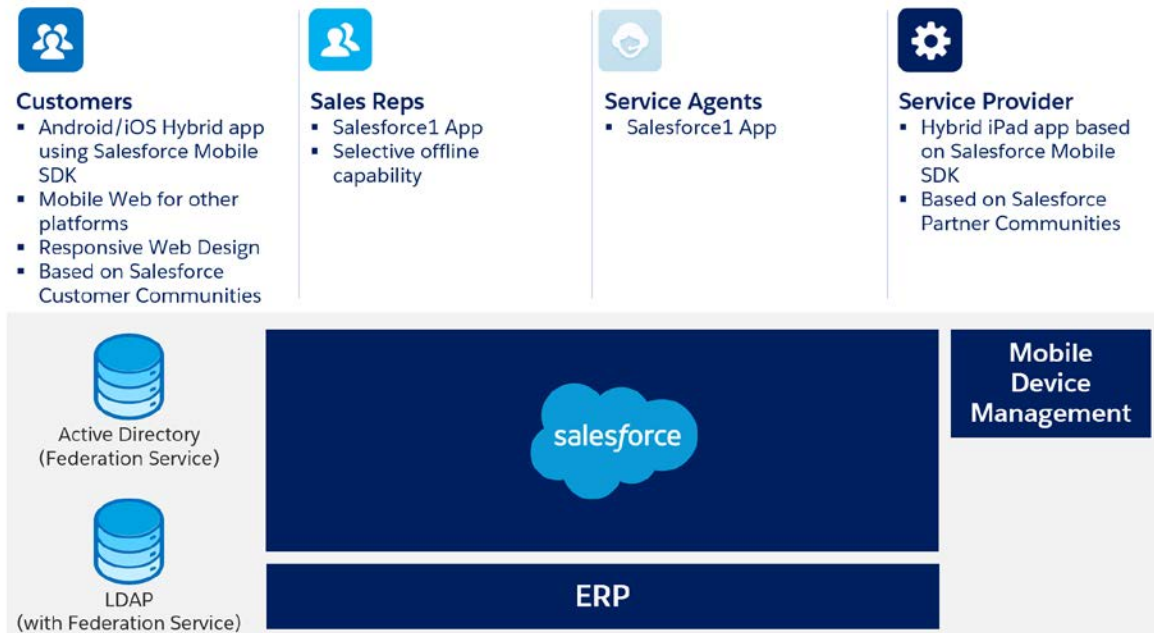
CONSIDERATIONS

1. Who are the key actors (user groups)? What are their specific requirements?
2. What assumptions can be made or what requirements can be inferred from the scenario? (For example, customer mobile development capability, security, offline storage, and desired user experience.)
3. What does near 100% market penetration mean for the potential solution? What platforms are the key players in the smartphone market, and what are their relative market shares?
4. What are the relative strengths of the Mobile Architectures / Application patterns (Mobile Web, Hybrid, Fully Native, Salesforce1)? How do their strengths apply to each requirement?



SOLUTION

Best Solution Overview



SOLUTION DISCUSSION:

SELF-SERVICE

- For customers, the recommended approach is to develop a hybrid mobile app using the Salesforce Mobile SDK for Android and iOS platforms:
 - These two platforms dominate the smartphone market; therefore maximum reach can be achieved.
 - The app can be distributed to customers using Google Play / Apple App Store.
 - The hybrid solution enables UW to leverage most of HTML5 development across platforms. The pages should be developed using a Responsive Web pattern, since it maximizes re-use across different mobile form factors, including tablet and desktop.
 - The remaining market can be reached using a Mobile Web solution.
- As customers want to be able to receive updates on their service request, a solution based on Salesforce Customer Communities is recommended.
 - This provides maximum extensibility in the future (e.g., view invoices, pay bills, etc.).



- The LDAP federated service can be set up as an authentication provider for Customers.
- If a solution without customers having to authenticate is required, then it's possible to develop a Force.com Sites page (mobile Responsive) and allow customers to capture their support cases via that page. If the customer provides their email, then notification to the cases can be provided via email.
- iOS / Android frameworks provide the ability to extend the functionality to Apple Watch / Android wear devices. These watch apps act as an "extension" of the phone app.
- Location Sharing: This can be enabled using the [HTML geolocation](#) feature where required.
- Uploading a Picture: The [HTML media capture](#) feature can be used in a hybrid or web app to enable the user to choose a picture on their phone to upload or to take a new picture.
- Alternatives:
 - Salesforce1: While Customer Community users can be enabled in. Salesforce1, this is not a great option to the mass market end users due to branding limitations and device coverage (iOS and Android only).
 - Fully Native: The fully native option provides the best user experience. However, it comes at a cost, as specialist skill sets are required to develop / maintain the app.

SALES TEAM:

- The Salesforce1 mobile app is the best solution for this user group.
 - Most of the functions they do on the road are simple form entry / data view.
 - They have standardized on iPads.
 - The Salesforce1 app provides the ability to access reports and dashboards out-of-the-box.
 - The Visualforce page for billing history / contracts can be mobile-enabled by ensuring it's responsive, and that the page is optimized for touch-based interaction.
 - The Salesforce1 mobile app provides limited offline access. For example, recent records can be viewed in offline mode.



- Alternatives:
 - If there are more sophisticated off-line requirements, for example, the ability to create / edit records, then a hybrid mobile solution based on Salesforce Mobile SDK might be required. This will be a significant investment increase on the Salesforce1 mobile app.

SERVICE AGENTS:

- The requirement to assign service requests based on the agent's proximity to the account will require a Custom-built hybrid / native app based on the Salesforce Mobile SDK:
 - This requirement implies a background process to periodically log the location of the agent to Salesforce, or an alternate platform such as Heroku if very frequent logging of the location is required.
 - A custom app is required to run a background process to enable this capability.
 - Alternative: If proximity-based routing is not a mandatory requirement, then a Salesforce1-based solution is a better fit for this user group (requires significantly less investment).
- BYOD support / security requirements: To meet the security requirements, the customer can adopt a Mobile Device Management (MDM) solution such as Mobile Iron or Airwatch. Security policies can be set up so that administrators can remotely wipe all data if a device is lost or stolen.
 - Some authentication providers can be enabled so that only devices enrolled in MDM can authenticate.
 - MDM policies can also be set up requiring a minimum device PIN code length / periodic expiry of PIN.

SERVICE PROVIDERS:

- A solution based on Partner Communities is best suited for this user group:
 - Touch Id integration and branding requirements (assumed) would mean a hybrid solution is best suited here.
 - The Touch Id integration can be enabled by using the appropriate iOS API (KeychainTouchID class).
 - Service Request visibility requirement can be enabled using the Salesforce record sharing capabilities; for example, Private OWD for cases. The REST APIs used by the mobile solution will respect the user's record visibility constraints set up in Salesforce.



- Alternative: If custom branding and Touch Id integration are mandatory requirements, then a Salesforce1-based solution is a better fit for this user group. It requires significantly less investment.

OTHER CONSIDERATIONS:

- Both Salesforce1 and Salesforce Mobile SDK apps can work with a SAML-based authentication provider (federated authentication).
 - Leverage AD with ADFS for internal users.
 - A Federation Service (for example Ping) is required for the LDAP authentication store for external users.
- Alternative: Provide a delegated authentication solution, but this is less desirable.



2. SALESFORCE1 MOBILE SOLUTION

Describe the design considerations, trade-offs, and risks for mobile solutions and recommend the appropriate mobile platform: HTML5, Native (iOS/android/Windows Mobile), Hybrid solutions, or Salesforce1 Mobile App.

OPPORTUNITY MANAGEMENT

A key requirement for the UW Sales Team is for their reps to have easy access to their opportunities from anywhere. These opportunities cover the various water products and services they offer to the market.

The reps always struggle for time, and therefore need to find ways to quickly create and/or update opportunities. The reps and, more importantly, their manager, also require the ability to track the progress of their opportunities individually and overall as a team.

Before an opportunity closes and progresses into a contract, reps need the ability to review its details even while on the road. This involves looking at a summarized view of the opportunity, including all the products related to it.

PREREQUISITE SETUP STEPS

- It is a prerequisite to have completed the “Mobile Architecture Design” build material.
- Access to a mobile device (smartphone or tablet).

CONSIDERATIONS

1. Who are your users? What are the key capabilities that they need access to on the road?
2. What are the declarative features available in implementing a Salesforce1 mobile app solution?
3. What are the various types of actions available for you to create declaratively in Salesforce?
4. What is the difference between a Global Quick Action and an Object-Specific Quick Action? When would you use one over the other?
5. What are the different approaches available in designing Visualforce pages for the Salesforce1 mobile application?



SOLUTION

SOLUTION DESCRIPTION

Configure the Salesforce1 app to meet the needs of the Sales Team. Go through Mobile Administration to manage navigation and other settings. Create a quick action and assign it to an appropriate layout. Develop a Visualforce page that provides a summarized view of an opportunity and all its products and display it on the Salesforce1 mobile app.

SOLUTION WALKTHROUGH

Manage Salesforce1 navigation through Mobile Administration. Improve productivity by moving more frequently used items higher in the menu. The Sales Team at UW use Reports a lot to track progress.

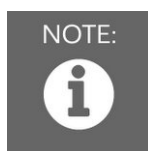
1. Adjust the order of Navigation Menu Items.

- Go to **Setup > Administer > Mobile Administration** and then go to **Salesforce1 Navigation**.
- Under the Selected panel, move **Reports** to be just above Dashboards using the arrows.
- Click **Save**.

2. Create an Object-Specific Action.

The following steps take you through the creation of an object-specific action and how to assign it to a layout to make it available in the Salesforce1 mobile app.

- Go to **Setup | Build | Customize**.
- In the object management settings for Opportunities, go to **Buttons, Links, and Actions**.
- Click **New Action**.
- For Action Type, select **Create a Record**.
- For Target Object, select **Opportunity Product**.
- For Label, enter New Opportunity Product.
- Click **Save**.



NOTE: *Leave the Standard Label Type as None.*



3. Assign the Action to a Page Layout.

- Navigate to the Page Layouts.
- Click **Edit** next to the Opportunity Layout.
- In the Salesforce1 and Lightning Experience Actions section, click **override the predefined actions**.
- Click the **Salesforce1 Actions** category in the palette and then drag the new action you created into the Salesforce1 and Lightning Experience Actions section.
- Click **Save**.

4. Develop a mobile-optimized Visualforce page.

To display data from Opportunity and Opportunity Product objects, create a Visualforce page and make it available on the Salesforce1 mobile app.

The recommended approach is to use JavaScript Remoting and Static HTML. This provides the best user experience, as it allows you to closely align the user interface to Salesforce1 while providing the most optimal performance.

- Install your preferred [Salesforce Mobile Pack](#) into your organization as a static resource.
- Set your page's docType to **html-5.0**. Strongly consider disabling the standard stylesheets and header.
- Add scripts and styles from your chosen mobile toolkit to the page using Visualforce resource tags.
- Use HTML5 and your mobile toolkit's tags and attributes to create a page skeleton.
- Add JavaScript functions to the page as handlers to respond to user interaction. Use JavaScript remoting to call Apex @RemoteAction methods that retrieve records, perform DML, and so on.
- Add additional JavaScript functions to handle user actions and page updates. Perform page updates by constructing HTML elements in JavaScript, and then adding or appending them to the page skeleton.

For more information on how to develop a Visualforce page suitable for the Salesforce1 mobile app while taking into account the relevant limitations, refer to the [Salesforce1 Mobile App Developer Guide](#), in particular the Development Guidelines and Best Practices section.



SOLUTIONS

1. Who are your users? What are the key capabilities that they need access to on the road?
 - a. The users in this scenario are Sales Team members (reps, managers, etc.), and they need to effectively manage opportunities and run reports/dashboards
2. What are the declarative features available in implementing a Salesforce1 mobile app solution?
 - a. Mobile Navigation, Layouts, Actions, and Branding, plus enabling Offline and Notifications.
3. What are the various types of actions available for you to create declaratively in Salesforce?
 - **Create actions** let users create records.
 - **Log a call actions** let users record the details of phone calls or other customer interactions.
 - **Question actions** enable users to ask and search for questions about the records that they're working with.
 - **Send email actions**, available only on Cases, give users access to a simplified version of the Case Feed **Email action** on Salesforce1.
 - **Update actions** let users make changes to a record.
4. What is the difference between a Global Quick Action and an Object-Specific Quick Action? When would you use one over the other?
 - a. Object-specific actions create records that are automatically associated with related records.
5. What are the different approaches available in designing Visualforce pages for the Salesforce1 mobile application?
 - a. The three (3) approaches available are:
 1. Standard Visualforce Pages
 2. Mixed Visualforce and HTML
 3. JavaScript Remoting and Static HTML

Standard Visualforce Pages would be the fastest to develop, but would have limitations around the visual design and do not really cater to a mobile-optimized user experience overall. Pages using Mixed Visualforce and HTML would be reasonably fast to develop and allow the look and feel to be closer to Salesforce1 through the use of CSS stylesheets. This approach, however, is still not fully optimal, since it follows Standard Visualforce in terms of request-response cycles, controller functionalities, etc. Finally, JavaScript Remoting and Static HTML provides the best in both user interface and performance, but takes the longest time to develop.



3. LIGHTNING DEVELOPMENT FOR SALESFORCE1

Recommend a mobile strategy, taking into consideration current Apex, Lightning Component, and Visualforce assets and extension to Salesforce1 declarative capabilities.

CONTRACT MANAGEMENT

A key requirement for the UW Sales Team is for their reps to create and submit new contracts while they are on the road. As part of that process, there is a multi-step Visualforce wizard which initiates a synchronous web service call to the ERP system on contract submission.

PREREQUISITE SETUP STEPS

- It is a prerequisite to have gone through Scenario 1: Mobile Architecture Design.
- It is a prerequisite to have gone through Scenario 2: Salesforce1 Mobile Solution in order to understand the declarative options available.
- Candidates should sign up for a [Developer Edition](#) (DE) or Practice org to use in building up the solution.
- Access to a Mobile device (smartphone or tablet) or an emulator.

CONSIDERATIONS

1. In this scenario, we are taking existing custom functionality in Salesforce Classic and porting it to Salesforce1. Mobile design and desktop design have major differences that need to be taken into consideration:
 - a. Minimum Viable Product (MVP): What are the core elements of the wizard that need to be accomplished in order for the contract submission to take place? What we are looking for here is a redux of the existing functionality to its essential pieces.
 - i. Can the number of fields be reduced?
 - ii. Can we reduce validation rules?
 - iii. Can we reduce pages?
 - b. A subset of MVP is Minimalist UX Design: As you design the pages in the Wizard you have to account for the fact that you have limited real estate and variable bandwidth. You have to design the page so it's simple to use with figure gestures and doesn't require a lot of scrolling.

These considerations are an ideal. In most cases, you end up with trade-offs, but rarely do you port a desktop design 1:1 to a mobile design.



2. Once you have a grasp of the basic design before moving to a custom solution in Salesforce1, you should consider if you can piece together the various declarative options available to achieve the design. A wizard in Visualforce on the desktop does not need to look or feel exactly the same in a mobile environment, and should take into consideration unique aspects of working on mobile devices as opposed to laptops/desktops. Ask questions that can possibly reduce complexity. For example:
 - a. Could you use a publisher action to achieve the desired result?
 - b. Could you make the submission to the ERP system Asynchronous and use Process Builder or Outbound Messaging to submit the request?

Try to go for the simplest solution that balances user experience, complexity, and performance.



SOLUTION

Customize the Salesforce1 app to meet the needs of the Sales Team by creating Lightning Components that can be leveraged for the Wizard functionality.

By using the Lightning, we will also be leveraging the Lightning Design System, which is the framework Salesforce uses to style its pages on all platforms: desktop and mobile. This will keep the look consistent with what other pages in Salesforce1 look like. Additionally, when you have the Lightning UI enabled on the desktop, the Mobile Wizard look and feel will be consistent with that look and feel. In fact, you can use the Lightning Component you develop here as a component in the desktop version, as well. The Lightning Design System is also “responsive.” This means it will adapt to different form factors (i.e., iPad, Desktop, iPhone) without having to create separate code or pages.

SOLUTION WALKTHROUGH

1. Determine the flow of the Wizard.

Assumption 1: We have done the work to minimize the wizard to what is absolutely necessary and workable for a mobile environment.

Assumption 2: Currently Lightning Components cannot be hooked up into the Publisher Action framework. For this reason, we will access the Wizard through the main left-hand navigation as opposed to alternative options such as on an Opportunity Record.

Assumption 3: Lightning Components requires that your org has been set up with a custom domain (My Domain feature).

Assumption 4: Lightning Components is enabled.

The Flow:

1. User will select **Contract Wizard** from the left-hand navigation in Salesforce1.
2. User will see a single page wizard.
3. User will search for the Opportunity for which they would like to create a Contract.
4. User will fill out some basic information for the Contract.
5. User will hit Submit Contract.
6. A synchronous call will be made to the third-party system.
7. A fail or success message will be returned.






What it'll look like:

The image shows a mobile application interface for a "Create Contract Wizard". At the top, there is a blue header bar with a hamburger menu icon on the left, and performance metrics "0.134 s" and "140.89 KB" along with search and notification icons on the right. Below the header is a dark blue title bar with the text "Create Contract Wizard". The main content area is divided into three steps: "STEP 1: SEARCH FOR OPPORTUNITY", "STEP 2: ENTER CONTRACT INFORMATION", and "STEP 3: CREATE CONTRACT". In Step 1, there is a text input field for "Opportunity Name" and a "Search" button. In Step 2, there is a "Name:" label followed by a "Contract Name" input field, and an "End Date:" label followed by an empty input field. In Step 3, there is a large "Create Contract" button.

Figure 1: The Wizard



0.134 s 140.89 KB

Pull to Refresh

Create Contract Wizard

STEP 1: SEARCH FOR OPPORTUNITY

Search

Express Logistics Portable Truck Generators

Express Logistics SLA

Express Logistics Standby Generator

End Date:

STEP 3: CREATE CONTRACT

Create Contract

Figure 2: Searching for an Opportunity



Create Contract Wizard

0.134 s 140.89 KB

STEP 1: SEARCH FOR OPPORTUNITY

Express

Express Logistics Portable Truck Generators

Express Logistics SLA

Express Logistics Standby Generator

End Date:

April 2016

SUN	MON	TUE	WED	THU	FRI	SAT
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

Today

STEP 3: CREATE CONTRACT

Create Contract

Figure 3: Date Picker to select the Contract End Date

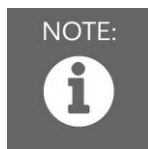
2. Create the Lightning Component.

1. We will be utilizing the Developer Console to create our Component.
 - a. Click **your name** on the upper right-hand side to access the drop-down menu.
 - b. Click **Developer Console**.
2. Create a Lightning Component.
 - a. Go to **File | New | Lightning Component**.
 - b. Name it **ContractWizard**.



3. Create a Lightning App as a test bed for your component.

1. Create a Lightning Application to test the Component on the desktop for efficiency. Lightning App is required to preview the component in a browser.
 - a. Open the Developer Console.
 - b. Click **File | New | Lightning Application**.
 - c. Name it OpptyContractWizard.app.
2. Add the following code:
 1. `<aura:application >`
 2. `<div class="slds">`
 3. `<ltng:require styles="/resource/sldsprod/assets/styles/salesforce-lightning-design-system-vf.css" />`
 4. `<c:ContractWizard />`
 5. `</div>`
 6. `</aura:application>`
3. While you have the Lightning App window active, you will see a Preview button on the upper right-hand side of the screen. You can click on this button to open a browser to view the component you are building.



NOTE:

Lightning Apps built in this way are not currently supported by Salesforce1. We will install the Component directly to Salesforce1 when we get to that step. Creating the Lightning App at this stage is just to facilitate testing.

4. Create your Wizard Component: Creating the UI.

1. Go back to your component you created in Step 2.
2. On the right-hand panel, you'll notice that you have the various components of a Lightning Component. For this tutorial, we will be primarily interested in the following:
 - a. Component
 - b. Controller



3. Make sure you are in the Component page (the title tab should end in .cmp) and enter the following code:

```
1 <aura:component controller="OpptyContractWizardApexController"
  implements="force:appHostable">
2   <aura:attribute name="oppty" type="Opportunity[]"/>
3
4   <div class="slds-box slds-theme--inverse" role="banner">
5     <div class="slds-media">
6       <div class="slds-media__body">
7         <p class="slds-page-header__title slds-truncate slds-align-middle"
  title="Rohde Corp - 80,000 Widgets">Create Contract Wizard</p>
8       </div>
9     </div>
10  </div>
11
12
13  <div class="slds-m-top--medium"></div>
14  <h3 class="slds-section-title--divider">Step 1: Search for Opportunity</h3>
1  <div class="slds-m-top--medium"></div>
15 <div class="slds-lookup" data-select="multi" data-scope="single" data-
  typeahead="true">
16   <div class="slds-form-element">
17     <div class="slds-form-element__control">
18       <div class="slds-form-element__control slds-input-has-icon slds-
  input-has-icon--right">
19         <ui:inputText label="" aura:recordId="" aura:id="opptyName"
  aura:aria-autocomplete="list" aura:role="combobox" aura:aria-
  expanded="true" aura:aria-activedescendant="" placeholder="Opportunity
  Name" class="slds-input"/>
20       </div>
21     </div>
22
23     <div class="slds-m-top--small"></div>
24     <div class="slds-text-align--right">
25       <ui:button label="Search" press="{!c.searchOppty}" class="slds-
  button--brand"/>
26     </div>
27
```



```
28     <div aura:id="opptyList" class="slds-lookup__menu slds-hide"
    aura:role="listbox">
29         <ul class="slds-lookup__list" aura:role="presentation">
30             <aura:iteration items="{!v.opptys}" var="oppty">
31                 <li class="slds-lookup__item">
32                     <a onclick="{!c.selectOppty}" id="{!oppty.Id}"
    aura:role="option">
33                         {!oppty.Name}
34
35                     </a>
36
37                 </li>
38             </aura:iteration>
39         </ul>
40     </div>
41
42 </div>
43 </div>
44
45 <div class="slds-m-top--large"></div>
46 <h3 class="slds-section-title--divider">Step 2: Enter Contract
    Information</h3>
47 <div class="slds-m-top--medium"></div>
48
49 <div class="slds-form-element">
50     <div class="slds-form-element__control">
51         <ui:inputText label="Name:" aura:id="contractName"
    placeholder="Contract Name" class="slds-input"/>
52     </div>
53     <div class="slds-m-top--medium"></div>
54     <div class="slds-form-element__control">
55         <ui:inputDate label="End Date:" aura:id="contractEndDate"
    class="slds-input"/>
56     </div>
57 </div>
58
59 <div class="slds-m-top--large"></div>
60 <h1 class="slds-section-title--divider">Step 3: Create Contract</h1>
61 <div class="slds-m-top--medium"></div>
62
63 <div class="slds-text-align--center">
```



```
64 <ui:button label="Create Contract" press="{!c.createContract}" class="slds-  
    button--brand"/>  
65 </div>  
66 <div class="slds-text-align--center">  
67     <ui:outputText aura:id="createContractOutput" value=""/>  
68  
69 </div>  
70  
71  
72 </aura:component>
```



Let's break this down so we can understand it better:

1. All components start with the `<aura:component>` tag.
 - To make this Component accessible to Salesforce1, we need to add to the tag the `implements="force:appHostable"` attribute/value.
 - If we are accessing any server-side logic, we can specify the corresponding class just like we do in a VisualForce page. In this case `controller="OpptyContractWizardApexController."` We will use this class to get the Opportunity records and create a Contract record later in the steps.
2. Another aspect of creating a Component is specifying attributes that you will reference in your page. Attributes are similar to variables. You use them to house values that you can manipulate or display within the page. In our case, `<aura:attribute name="opptys" type="Opportunity[]"/>` will hold a collection of Opportunities based on the search term provided by the user. We will iterate this collection further down in the page. More info: https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_attributes.htm
3. Next, we create our form to collect the information we need to create the contract.
 - Visuals: We will be utilizing the Lightning Design System (LDS) to create the look and feel for our page. As of the latest version of Lightning Components, you don't need to do anything additional to leverage LDS. It's loaded as part of the Framework. The structure of the HTML is governed by what is specified for the various elements in the Lightning Design System site: <https://www.lightningdesignsystem.com/>
 - The Lightning Framework comes with UI elements that will help you save time/effort. These tags begin with "ui." We're using:
 - `ui:button`
 - `ui:inputText`
 - `ui:outputText`
 - `ui:inputDate`
 - Launches the calendar in iOS for easier data entry.
 - For a full listing of what elements are available and what types of functionality and events they support, go to: <https://<myDomain>.lightning.force.com/auradocs/reference.app>, where `<myDomain>` is the name of your custom Salesforce domain.



- The Lightning Framework also allows you to do some basic conditional and iterative logic to manipulate the presentation of the page similar to VisualForce. In this case, we are using `<aura:iteration />` to iterate through the attribute we defined to house the collection of Opportunities returned to us by our search query.
- 4. As mentioned earlier, we are taking our look and feel from LDS. For searching for the Opportunity records, the user enters what they know of the Opportunity Name and clicks search. When they click search, a drop-down appears with the search results. When they click on a search result item, the item's record id and name are recorded and shown in the original Search Text box. The drop-down then disappears. To achieve this effect, we need to dynamically show and hide the drop-down. The Lightning Framework allows you to do this in a special way using the code below. More on this when we discuss the controller logic.
 - `$A.util.removeClass(divSearchList, 'slds-show');`
 - `$A.util.addClass(divSearchList, 'slds-hide');`
- 5. Create your Wizard Component: Creating the Client-Side Controller Logic.
 1. While you're in the the Component page for your Component, double-click on the Controller page on the right-hand panel to access the page where we will write the Controller Logic.
 2. Paste the following code into the Controller Page:

```
1  ({
2
3  "searchOpptys" : function(cmp) {
4  var action = cmp.get("c.findOppty");
5
6  action.setParams({ searchKey : cmp.find("opptyName").get("v.value")
7  });
8  action.setCallback(this, function(response) {
9  var state = response.getState();
10
11  if (state === "SUCCESS") {
12  var divSearchList = cmp.find("opptyList");
13  $A.util.removeClass(divSearchList, 'slds-hide');
14  $A.util.addClass(divSearchList, 'slds-show');
15
16  cmp.set("v.opptys", response.getReturnValue());
```



```
17 }
18 else if (state === "INCOMPLETE") {
19 // do something
20 }
21 else if (state === "ERROR") {
22 var errors = response.getError();
23 if (errors) {
24 if (errors[0] && errors[0].message) {
25 console.log("Error message: " +
26 errors[0].message);
27 }
28 } else {
29 console.log("Unknown error");
30 }
31 }
32 });
33
34
35 $A.enqueueAction(action);
36 },
37
38 "selectOppty" : function(cmp, event) {
39
40 var attributeValue = cmp.get("v.text");
41 var divSearchList = cmp.find("opptyList");
42 var opptyNameInput = cmp.find("opptyName");
43 var target;
44
45 opptyNameInput.set("v.value", event.target.innerHTML);
46 opptyNameInput.set("v.recordId", event.target.id);
47
48 $A.util.removeClass(divSearchList, 'slds-show');
49 $A.util.addClass(divSearchList, 'slds-hide');
50
51 },
52
53 "createContract" : function(cmp, event) {
54
55 var action = cmp.get("c.createContract");
56 var returnMsg = cmp.find("createContractOutput");
```



```
57
58
59 action.setParams({ OpportunityId :
    cmp.find("opptyName").get("v.recordId") });
60 action.setParams({ ContractName :
    cmp.find("contractName").get("v.value") });
61 action.setParams({ ContractEndDate :
    cmp.find("contractEndDate").get("v.value") });
62
63 action.setCallback(this, function(response) {
64 var state = response.getState();
65
66 if (state === "SUCCESS") {
67
68 returnMsg.set("v.value", response.getReturnValue());
69 }
70 else if (state === "INCOMPLETE") {
71 // do something
72 }
73 else if (state === "ERROR") {
74 var errors = response.getError();
75 if (errors) {
76 if (errors[0] && errors[0].message) {
77 console.log("Error message: " +
78 errors[0].message);
79 }
80 } else {
81 console.log("Unknown error");
82 }
83 }
84 });
85
86
87 $A.enqueueAction(action);
88
89 }
90 })
```



3. The controller page handles three functions for us:
 - a. searchOpptys - Will call the server-side controller class to fetch the search results. The function is triggered on the press event of the Search button. The code is pretty straightforward, so we won't get into too much detail, except for mentioning how you reference objects in the Component page:
 - i. Every function as a reference to the Component page. This is typically passed in with the name "cmp."
 - ii. You use the find method to find an object on the Component page. For example, if I want to find the inputText field with the id "opptyName," I would write `cmp.find("opptyName")`.
 - iii. You get and set values using the get and set methods. You reference the attribute within an Object using `v.[attribute]` notation. For example, to get the label value for the opptyName inputText field, I would write `cmp.find("opptyName").get("v.label")`.
 - iv. You can define custom attributes, but make sure they have the aura: prefix. For example, `<ui:inputText aura:myCustomAttribute="value">`.
 - v. You'll also notice in this function that I'm modifying the style values for the drop-down div to hide/show. Again this is done by using the `$A.util.removeClass` and `$A.util.addClass` functions.
 - b. selectOppty - This function takes the value selected from the search drop-down list and populates the Opportunity ID and Opportunity name to the opptyName inputText field. Additionally, it calls the `$A.util` class functions to hide the drop-down after a search item is selected. The Opportunity ID is stored in a custom attributed (`aura:recordId`).
 - c. createContract - Calls the create contract method on the server-side controller and passes the field values. Returns a success message upon the completion of the call.



6. Create the Server-Side Controller.

1. Create a new class in the Developer Console:
 - **File | New | Apex Class.**
 - Call it OpptyContractWizardApexController.
2. Paste the following code:
 1. public with sharing class OpptyContractWizardApexController {
 - 2.
 3. @AuraEnabled
 4. public static List<Opportunity> findOppty(String searchKey)
 5. {
 6. String srchTerm = '%' + searchKey + '%';
 7. return [select id, name FROM Opportunity where name LIKE :srchTerm
 8. LIMIT 5];
 9. }
 - 10.
 11. @AuraEnabled
 12. public static String createContract(String OpportunityId, String
 13. ContractName, String ContractEndDate)
 14. {
 15. //Add logic to create contract and submit to ERP system.
 - 16.
 17. return 'Contract Successfully Created';
 18. }
 19. }

Let's break this down again:

1. First things first - don't forget about security. Use the "With Sharing" keywords if you want to limit scope to what records users have access to.
2. Defining a server-side method class to be accessible to a Lightning Component:
 - a. a. Use the @AuraEnabled annotation above the method name.
 - b. b. Define the method as public static.



3. The rest is typical Apex coding we are all familiar with.
 - a. findOppty function: Gets a list of Opportunities based on the search term entered on the form. I've limited it to just the top 5. You can add more, but remember on mobile devices more is not better!
 - b. createContract function: I've left this function with just a placeholder as the remaining work is typical Apex coding:
 - Using the Opportunity record, you can get the additional information required to create the Contract record, such as the Account ID.
 - Once you have all the information, create the Contract record using standard Apex Coding.
 - You can make a synchronous call to the ERP system or handle it through asynchronous methods, such as Outbound Messaging, to deal with connectivity concerns if the ERP calls are costly.

7. Test the Component using the Lightning App Preview functionality.

- 1 Handy Tip: You can use Chrome Developer tools that come with the Chrome Browser to better debug client-side Javascript and HTML coding issues.

8. Add it to Salesforce1.

- 1 Go to **Setup | Customize | Create | Tabs**.
 - Add the component we created in the Lightning Components Tab Section.
- 2 Go to **Setup | Administer | Mobile Administration | Salesforce1 Navigation**.
 - Add the Lightning Component Tab to the Salesforce1 Navigation.

9. Check it Out!



CONSIDERATIONS FOR RECOMMENDED SOLUTION

The solution example demonstrates all the fundamental concepts of creating a Lightning Component for Salesforce1:

- 1 How to create HTML and dynamic HTML.
- 2 Examples of how to leverage the Lightning Design System to make your UX work faster / simpler.
- 3 How to handle client-side events.
- 4 How to do DML operations using server-side logic.

When it comes to designing for mobile, performance is the most important factor in usability, so getting the right balance between performance and creating a rich and functional user experience should be the ultimate goal.

To expand on this example, there are some additional considerations not touched upon:

- 1 Error Handling:
 - a. We didn't go into validation rules; however, when a record is created all the validation rules set up in Salesforce will be enforced. The server-side method should return the error message from the resulting DML operation attempt and pass it to the Lightning Component so the user can view it. LDS has several sections devoted to the right way to display error messages.
 - b. There is some debate on how much to do this pro-actively. The debate primarily centers around how much complexity you build into the JavaScript code in addition to all the stuff you've already configured on the server-side. The benefit of doing it pro-actively is that it avoids user frustration as you can ideally do it at the point of data entry. There's some low hanging fruit by taking advantage of the Placeholder attribute, providing visual indicators for required fields, and help text (check out the LDS for this as well). If you use the "ui" form element tags, there is some built-in validation to make sure the user enters the expect text - for example, date for the ui:inputTag element.
 - c. The remote call to the ERP should have some built-in logic if the call is taking too long or fails. There are various integration patterns to handle this, such as queuing, but beyond the scope of this tutorial.



2. Publisher Actions versus Lightning Components: One current limitation of Lightning Components is that they cannot be tied to a Publisher Action when viewing a record but, instead, must be accessed through the left-hand navigation. In this use case it may be more ideal if a user can directly select from the Opportunity record the ability to create a contract, as opposed to searching for the Opportunity in our Wizard. As Visualforce is supported for Publisher Actions, a work-around for this would be to create our Wizard in Visualforce. However, standard Visualforce coding isn't optimal for mobile situations given the inherent chattiness of Visualforce and the additional payload heft created by the viewstate. To get around these issues, there are mobile-optimized ways to create a Visualforce page using similar concepts as Lightning Components, including using LDS. It's beyond the scope of this tutorial to get into these optimizations.
3. Is the UI we created the most optimal? There are many ways to create the UI by mixing and matching various elements of the LDS. This is just one example. Try your own out. For example, instead of creating a search button, I could create a button within the text field itself!
4. Lightning Component framework is right now rapidly iterating, as is the Lightning Design System. There are many tweaks and changes. It may make sense given where we are in the maturity cycle to limit the use of Lightning Components for only the use cases where they make the most sense (i.e., you're using Lightning on the Desktop and want leverage the same component on the Desktop and Mobile) until the framework stabilizes. As an interim measure, you could start converting your VF pages to "mobile-optimized" VF pages using LDS. If done right, this could also be a middle step before you have to completely convert your existing VF pages to Lightning Components, as you should be able to leverage a lot of the client-side code/HTML mark.



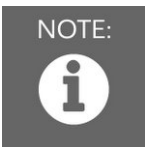
ADDITIONAL ACTIVITIES

INSTALL THE EXPENSE TRACKER APP

To work with Lightning apps and components, follow these prerequisites:

- Create a Developer Edition organization
- Register a Namespace Prefix
- Enable Lightning Components in Salesforce1

A package is a bundle of components that you can install in your org. This packaged app is useful if you want to learn about the Lightning app.



NOTE:

Install the package in an org that doesn't have any of the objects with the same API name as the quick start objects.

To install the Expense Tracker app:

1. Click the installation URL [link](#)
2. Log in to your organization by entering your username and password.
3. On the Package Installation Details page, click **Continue**.
4. Click **Next**, and on the Security Level page click **Next**.
5. Click **Install**.
6. Click **Deploy Now** and then **Deploy**.

When the installation completes, you can select the **Expenses** tab on the user interface to add new expense records.

You'll also see the Expenses menu item on the Salesforce1 navigation menu. If you don't see the menu item in Salesforce1, add it by going to **Mobile Administration** > **Mobile Navigation**.

Next, you can modify the code in the Developer Console or explore the standalone app at `https://<mySalesforceInstance>.lightning.force.com/c/expenseTracker.app`, where `<mySalesforceInstance>` is the name of the instance hosting your org; for example, `na1`.



CREATE AN EXPENSE OBJECT

Create an expense object to store your expense records and data for the app.

- 1 From Setup, click **Create > Objects**.
- 2 Click **New Custom Object**.
- 3 Fill in the custom object definition.
 - For the Label, enter Expense.
 - For the Plural Label, enter Expenses.
- 4 Click **Save** to finish creating your new object. The Expense detail page is displayed.

NOTE:



NOTE:

If you're using a namespace prefix, you might see namespace__Expense__c instead of Expense__c.

- 5 On the Expense detail page, add the following custom fields.

Field Type	Field Label
Number (16, 2)	Amount
Text (20)	Client
Date/Time	Date
Checkbox	Reimbursed?

To continue with this activity, please continue the steps found on page 11 of the [Lightning Components Developer's Guide](#).