Revolent Tech Task – Bronze

Imperative APEX Calls with Lightning Web Components

Scenario

This exercise explores various concepts within Lightning Web Components such as datatable, Apex controller and imperative Apex calls from Javascript. The objective is to create a Lightning Web Component that can be used for filtering a list of Leads based on the Lead status. The user will be able to click a button to select the desired Lead status and the component should render the list accordingly.

Step 1: Install necessary components

Install the following components on your system:

- Visual Studio Code
- Salesforce CLI
- Salesforce Extension Pack

Step 2: Project setup and org authorization

Once the components from Step 1 are installed, open VS Code. Hit Ctrl+Shift+P to open up the command palette. Run the command SFDX: Create Project. Accept the standard template. For the project name, enter *lwcPractice* and press **Enter**. Select a location for your project and click **Create Project**.

Once the project is created, open the command palette again and run the command SFDX: Authorize an Org. Press **Enter** to select **Project Default**. Then press **Enter** to select the default alias. The Salesforce login page should open up in a new browser window. Log in with your username and password. If prompted to allow access, click **Allow**. Now your org should be connected to your project for any future deployments.

Right-click on the *lwc* folder (under *force/app/default*) and click *SFDX: Create Lightning Web Component*. Alternatively, one can run this command from the command palette. Enter *leadsFilter* for the component name. Press **Enter** to select the default *force-app/main/default/lwc*. Press **Enter**. The new Lightning Web Component will now be present under the *lwc* folder.

Step 3: Apex controller

Right-click on the *classes* folder (under under *force/app/default*) and select **SFDX: Create Apex Class**. Alternatively, one can run this command from the command palette. For the class name, enter *LeadsController*. Press **Enter** to select the default location. Inside the apex class, create 5 methods. Each method should return a list of leads via a SOQL query.

Method Name	Fields to Query	Query Filter	Order By
getAllLeads	Name, Company, Email, Phone, Status	none	Name
getOpenNotContactedLeads	Name, Company, Email, Phone, Status	Status = 'Open - Not Contacted'	Name
getWorkingContactedLeads	Name, Company, Email, Phone, Status	Status = 'Working - Contacted'	Name
getClosedConvertedLeads	Name, Company, Email, Phone, Status	Status = 'Closed - Converted'	Name
getClosedNotConvertedLeads	Name, Company, Email, Phone, Status	Status = 'Closed - Not Converted'	Name

Enable object and field-level permissions on each SOQL query.

Step 4: HTML template

Inside the HTML template, create a lightning card component with the title Lead Filter and use custom101 as the icon. The template should also contain 5 lightning buttons with the labels **All**, **Open – Not Contacted**, **Working Contacted**, **Closed – Converted** and **Closed – Not Converted**. Each button should have onclick events that will call the corresponding event handler in the JavaScript file. Lastly, the template should contain a lightning datatable component that should conditionally display a table of Leads (if Leads data from the JavaScript file is available). The columns to be displayed should be specified in the JavaScript file.

```
<template>
    <!--TODO 1: Add required attributes to the lightning-card component-->
    <lightning-card>
        <div class="slds-m-around_medium">


            <p class="slds-text-heading_medium">Select a status</p>


            <br>


            <!--TODO 2: Create 5 lightning-button components-->


            <!--TODO 3: Create a lightning-datatable-->


        </div>
    </lightning-card>
</template>
```

Step 5: JavaScript File

In the JavaScript file, import the methods from the Apex controller and the following fields from Salesforce schema:

- Name
- Company
- Email
- Phone
- Status

Use the field imports to specify the columns to be used for the datatable in the HTML template.

Create 5 event handlers, one for each method in the Apex controller. Ensure that any error in the event handlers are caught.

```
//TODO 4: Necessary imports


const COLUMNS = [//TODO 5: Specify the fields to be used for the columns];


export default class LeadsFilter extends LightningElement {
    columns = COLUMNS;
    leads;
    error;


    //TODO 6: Create an event handler for each Apex method
}
```

Step 6: Config File

Replace the contents of the config file (ends with .js-meta.xml) with the following so that the Lightning Web Component is available on app pages in an org.

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

Step 7: Deployment

Save your files and deploy the lightning web component by right-clicking on the *force-app/main/default* folder and selecting **SFDX: Deploy Source to Org**. Alternatively, one can run this command from the command palette.

Step 8: Create Lightning Page

In your Salesforce org, go to **Setup** > **Lightning App Builder**. Create a new lightning page:

a. Click **New**

b. Choose **App Page**. Click **Next**.
c. For the label, enter **DemoPage**. Then click **Next**.
d. Select **Header and Left Sidebar** for the layout
e. Click **Finish**

Drag the *leadsFilter* component to the top region. Save and activate your page. You can now find the new lightning page in the app launcher.