77. Basics of Lightning Component and LDS- 27 July 2022

1] HelloWorld Aura Lightning Component

2] Structure of Aura Component

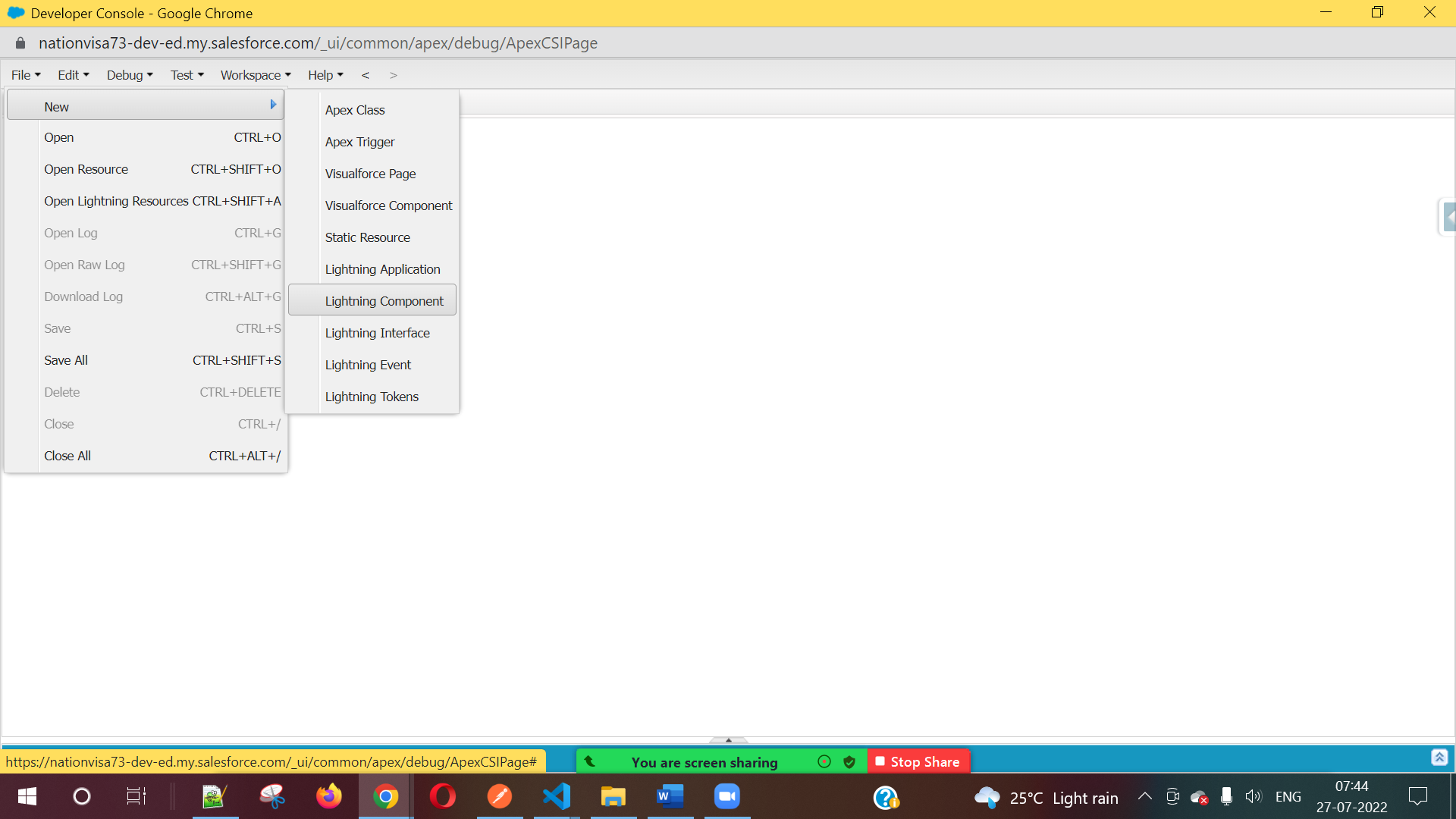
3] LDS

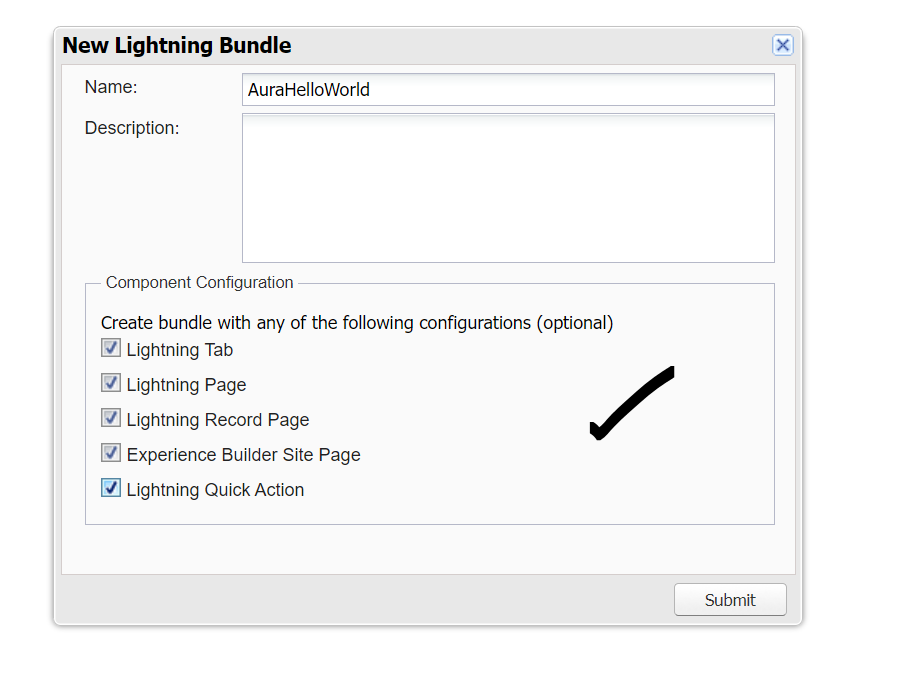
Aura Lightning component is a reusable component which is used to write the custom logic and it provides very good UI (User Interface).

OR

The Lightning Component framework is **a UI framework for developing single page applications for mobile and desktop devices**.

1] HelloWorld Aura Lightning Component





<aura:component implements="force:appHostable,flexipage:availableForAllPageTypes,flexipage:availableForRecordHome,force:hasRecordId,forceCommunity:availableForAllPageTypes,force:lightningQuickAction" access="global" >

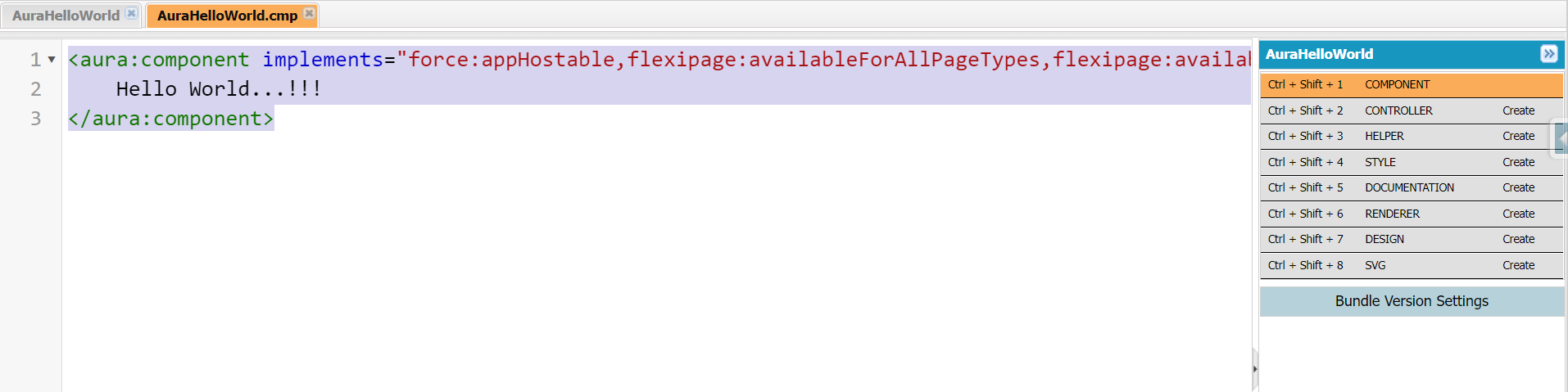
Hello World...!!!

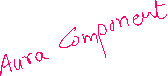
</aura:component>



2] Structure of Aura Component







Interfaces

Mostly used Lightning Interface

1) force:appHostable 🡺 Tab

2) flexipage:availableForAllPageTypes 🡺 App Page

3) flexipage:availableForRecordHome 🡺 Details Page

4) force:lightningQuickAction 🡺 QuickAction Button

5) forceCommunity:availableForAllPageTypes 🡺 Community

6) force:hasRecordId 🡺 to get Current Record Id

Bundle: <https://bestirtech.com/blog/2020/12/lightning-component-bundle-in-salesforce/>

**Resources of a lightning component bundle**

* + Component
  + Controller
  + Helper
  + Style
  + Documentation
  + Renderer
  + Design
  + SVG

**Component**

Contains the Html like code for a component along with some lightning tags. Whenever you start building your component code, the root tag for the component will be <aura: component> tag and you can design the component using other available aura tags.

**Controller**

Contains the javascript code for your component to call the action events declared in the lightning component. It calls the doInit method that runs before your component Html content is loaded, doInit is not required in all components.

**Helper**

Helper contains the javascript code for your component just like your controller but the purpose of the helper is to segregate the calls to the server. The methods defined inside the helper can be called from the controller and renderer, hence making it more reusable. You write those methods in the helper which are basically reusable and may be required to be called multiple times. So it is best practice to separate your client-side javascript from your server calls as the server calls run asynchronously.

Helper functions are local to a component, improve code reuse, and move the heavy lifting of JavaScript logic away from the client-side controller, wherever possible.

* 1. Use Controllers to listen to user events and other events like components, application events. But delegate business logic to the helper.
  2. Do similar delegation in all Renderer functions (render, re-render, and so on).
  3. Anytime you need to call one controller function from another controller function, move that logic to Helper.

**Style**

This style resource contains the custom style classes that you are going to use in your component.

**Renderer**

Contains code that you want to execute before or after rendering of your component elements. You can make calls to your helper methods also in the renderer. There are four types of rendering behavior i.e. Render, AfterRender, Rerender, and Unrender.

**Render**: Render function is called when the component is being initialized. This function returns an array of DOM nodes. Use render function to modify DOM or to modify component markup.

**AfterRender**: AfterRender function is called by the framework after the render function. AfterRender function allows you to further change an already rendered component. The afterRender function enables you to interact with the DOM tree after the framework’s rendering service has inserted DOM elements.

**Rerender**: Rerender function will be called when a component’s value changes due to user action like a button click or some other event like component/application event. Rerender function updates component upon re-rendering of the component.

**Unrender**: Framework fires an unrender event when a component is deleted. The base unrender function deletes all the DOM nodes rendered by a component’s render function. It is called by the framework when a component is being destroyed. Customize this behavior by overriding unrender in your component’s renderer. This can be useful when you are working with third-party libraries that are not native to the framework.

**Design**

Design resource is used to create attributes that you want to use at the design time i.e. while using the Lightning App builder tool. When you are using your components on a record detail page you have the option to set some of the attributes for your components to customize their behavior. These attributes are specified in the design component.

**SVG**

SVG stands for Scalable Vector Graphics. It is a custom icon resource for components used in Lightning App Builder or Community Builder and if we want to customize this icon to some other icon, we need to create an SVG for our lightning component. so that, when viewing the component at design time(in Lightning App Builder) the component appears with the icon specified in the SVG component.

Aura Component Variable:

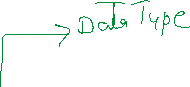


Apex:

Integer x=100;



System.debug(x);







<aura:component implements="force:appHostable,flexipage:availableForAllPageTypes,flexipage:availableForRecordHome,force:hasRecordId,forceCommunity:availableForAllPageTypes,force:lightningQuickAction" access="global" >

<aura:attribute name="num" type="Integer" default="100"/>

<aura:attribute name="myName" type="String" default="Silver Microsystems"/>

<aura:attribute name="amount" type="Decimal" default="300.90" />

<aura:attribute name="flag" type="Boolean" default="true"/>

<aura:attribute name="accountObject" type="Account" default="{'sObjectType' : 'Account'}" />

<lightning:card title="Hello">

Print = {!v.num} <br/>

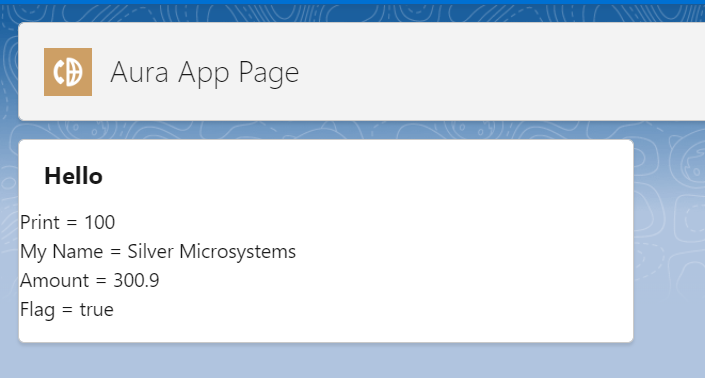
My Name = {!v.myName}<br/>

Amount = {!v.amount} <br />

Flag = {!v.flag}

</lightning:card>

</aura:component>



LDS : Lightning Data Services

LINK : <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/data_service.htm>

1] Create a new Account

<aura:component implements="force:appHostable,flexipage:availableForAllPageTypes,flexipage:availableForRecordHome,force:hasRecordId,forceCommunity:availableForAllPageTypes,force:lightningQuickAction" access="global" >

<aura:attribute name="fields" type="String[]" default="['Name', 'Industry', 'SLA\_\_c' , ‘ParentId’]"/>

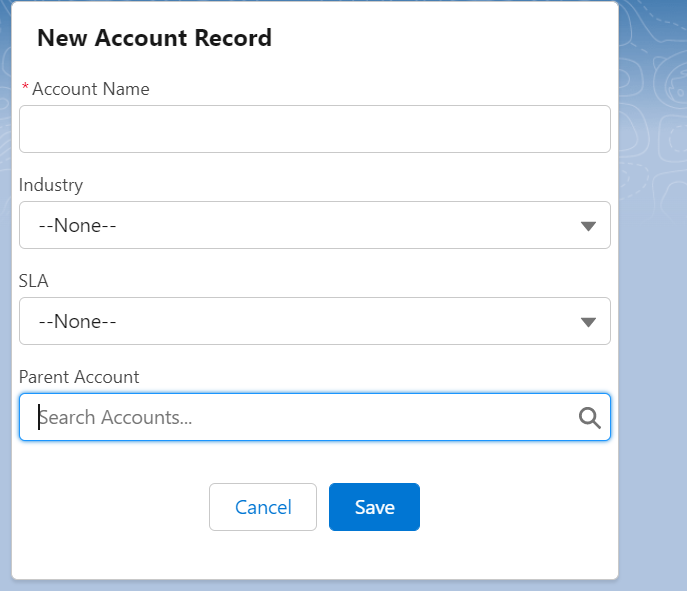
<lightning:card title="New Account Record">

<lightning:recordForm objectApiName="Account"

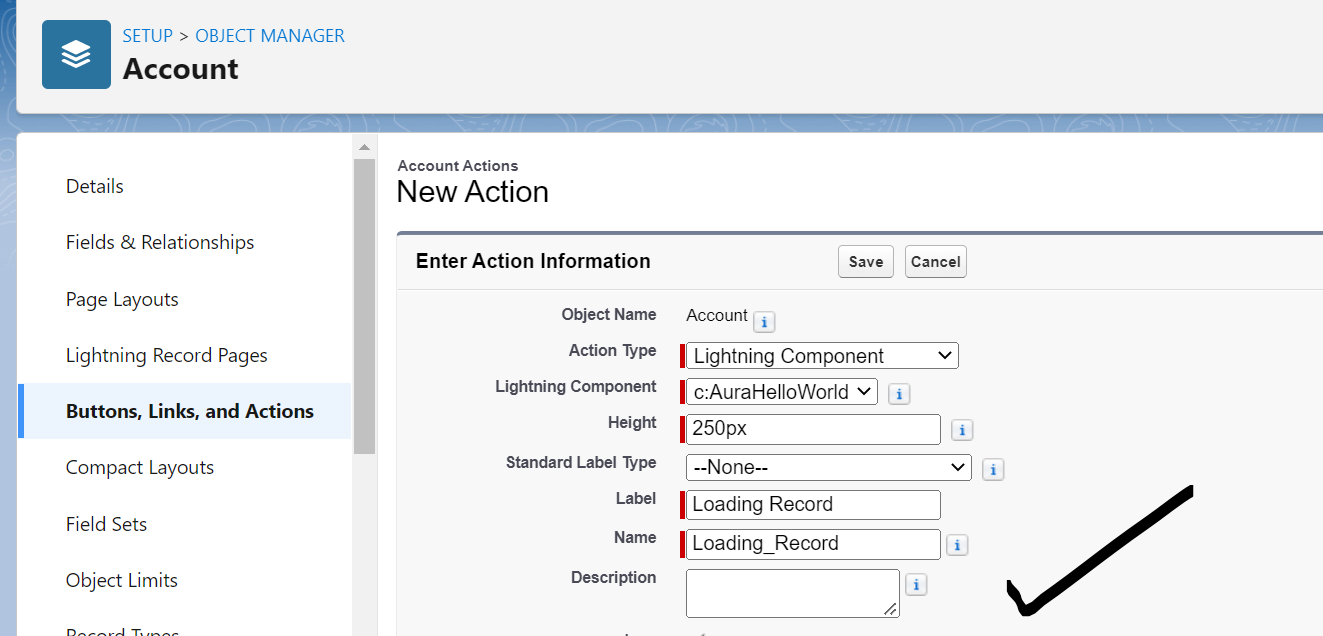
fields="{!v.fields}"/>

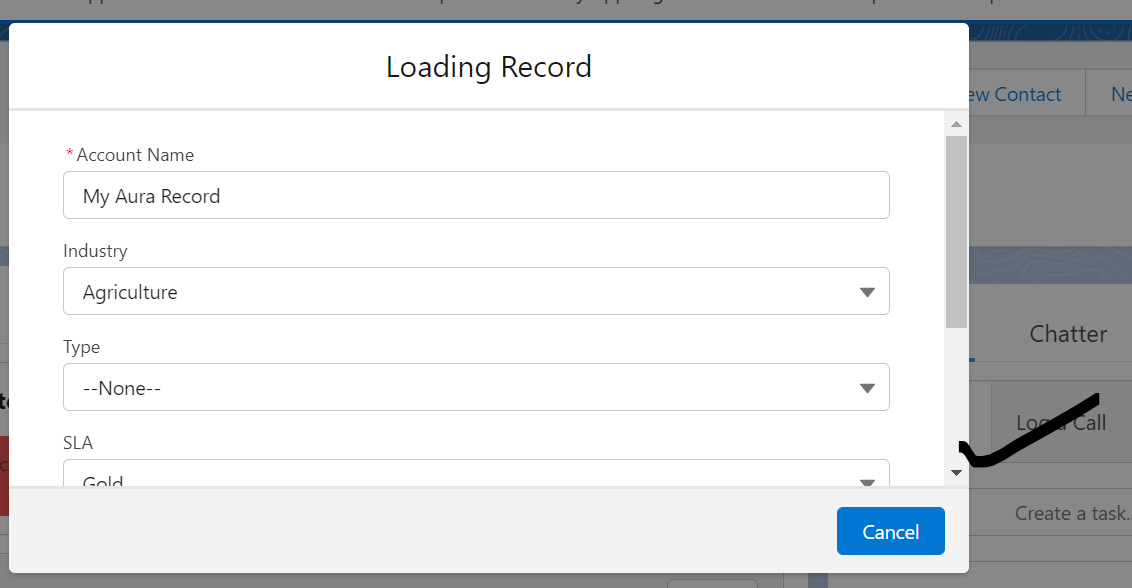
</lightning:card>

</aura:component>



2] Load Account Record and Edit a Record.





<aura:component implements="force:appHostable,flexipage:availableForAllPageTypes,flexipage:availableForRecordHome,force:hasRecordId,forceCommunity:availableForAllPageTypes,force:lightningQuickAction" access="global" >

<aura:attribute name="fields" type="String[]" default="['Name','Industry', 'Type', 'SLA\_\_c', 'ParentId']" />

<lightning:recordForm recordId="{!v.recordId}"

objectApiName="Account"

mode="edit"

fields="{!v.fields}" />

</aura:component>