Best Practice Coding Standards - LWC

1. **Naming convention**
   a. LWC component Bundle
      Html file : Use camel case to name your component and use kebab-case to reference a component in the markup

      JavaScript File : Java Script Class name should be in PascalCase

      Bundle Component : use camelCase.
   b. Custom Event Names

      Avoid prefixing custom event names with ‘on’. The event handler names start with on. So, adding an additional ‘on’ at the start of the event name would create confusion.

   Use below:

```
const customEvt = new CustomEvent('select');
```

2. Modular Components Approach:

a) Create and use reusable Lightning Web Components

b) Component Composition

Components are the building blocks of a page. We should consider the following point before dividing the components in LWC

Level 1: Based on its functionality

Level 2: Based on its reusability

When there is need to pass lots of inputs to the component, consider passing objects as parameters instead of having multiple attributes.

c) Create multiple child components for main component to increase the readability/debugging/maintainability.

3. Commenting:

- All methods/functions should be properly commented. The comment should describe what exactly the function is doing, what are the input parameters and returning values.
- Do not skip comments to make file smaller.

4. Multiple js files/lwc components should be used instead of a single big component or complex functionality

5. Variable declaration should be at same section. It should not be scattered all over the org.

6. import statements should not be too much in the js file

7. There should be no repetition of the code. If any exists, then move the code to new function/method and use the function.

8. Indentation

There should be proper indentation of the code having the spaces/formatting/line breaks as needed to increase the readability.

9. Extract utility methods outside your LWC components as LWC components should only deal with UI-related logic.

10. Include error handling in the code logic

Include error handling in the code logic: We should consider the error scenario as well and notify the user with a user friendly error message. For this, we can make use of toast message to show high level detail to user on what went wrong.

11. Avoid using hardcoding, use custom labels.

Sample Code:

1. lwc_Standards.html
2. lwc_Standards.js
3. util.js

1. **lwc_Standards.html**

```
<template>
        <!--Create a responsive layout  -->
<div class="c-container">

        <!--Responsive layout details of what UI it is displaying.
        Comments START -->
        <lightning-layout multiple-rows="true">

                <lightning-layout-item padding="around-small" size="
12">
                        <div class="page-section page-header">
                                <h2><strong style="color: #ff8000;"
>Header Section</strong></h2>
                        </div>
                </lightning-layout-item>

                <lightning-layout-item padding="around-small" size="
12">
                        <lightning-layout>
                                <lightning-layout-item padding="
around-small" size="3">
                                        <div class="page-section page-
right">
                                                <h2><strong style="
color: #ff8000;">Left Sidebar</strong></h2>
                                                <p> </p>
                                        </div>
                                </lightning-layout-item>
                                <lightning-layout-item padding="
around-small" size="6">
                                        <div class="page-section page-
main">
                                                <h2><strong style="
color: #ff8000;">Main Container</strong></h2>
                                                <p></p>

                                        </div>
                                </lightning-layout-item>
                                <lightning-layout-item padding="
around-small" size="3">
                                        <div class="page-section page-
right">
                                          <h2><strong style="color:
#ff8000;">Right Sidebar</strong></h2>

                                        <div class="slds-nav-
vertical__section"> </div>
                                        </div>
                                </lightning-layout-item>
                        </lightning-layout>
```

```
                </lightning-layout-item>
                <lightning-layout-item flexibility="auto" padding="
around-small" size="12">
                        <div class="page-footer page-section">
                                <h2><strong style="color: #ff8000;"
>Footer</strong></h2>
                        </div>
                </lightning-layout-item>
        </lightning-layout>
        <!--Responsive layout Comments END-->
</div>


</template>
```

2. lwc_Standards.js

```
// import Standard Library
import { LightningElement } from 'lwc';
// import show toast message
import { CurrentPageReference } from 'lightning/navigation';
// import navigation
import { NavigationMixin } from 'lightning/navigation';
// import Static Resources
import myResource from '@salesforce/resourceUrl
/namespace__resourceReference';
// import Custom Labels
import HomePageLabel from '@salesforce/label/c.HomePageNewsLabel';
// import apex class
import apexMethod from '@salesforce/apex/Namespace.Classname.
apexMethod';
// import util module
import {className,MethodName,VariableName,CustomLabel} from 'c/util';

export default class LwcPageLayout extends NavigationMixin
(LightningElement) {

        // declare and initialize  static resources variable value
         const img = myResource;
        // declare and initialization custom variable name
         clabel = clabel;
        // declare Global variable
        @api name;
        // declare boolean variable
        flag =true;
        // declare array
        arrName =[];
```

```
        // declare json object
        jsonList ={};
        // declare variable with initialization
        x =10;
        // declare variable without initializtion
        prodName;

        // wire method
        @wire(CurrentPageReference) ageRef;

        // connected callback for default value initialization
        connectedCallback(){
                // code here
        }

        // renderedCallback for  re-rendered when the value changes
        renderedCallback(){
                // code here
        }

        // define custom methods
        display(){
                //code here
        }

}
```

3. util.js

```
// import Standard Library
// import Static Resources
import myResource from '@salesforce/resourceUrl
/namespace__resourceReference';
// import Custom Labels
import HomePageLabel from '@salesforce/label/c.HomePageNewsLabel';
// import apex class
import apexMethod from '@salesforce/apex/Namespace.Classname.
apexMethod';

// variable declaration
        name;
        listOfContact =[];

// define custom labels
        clabel='';

// declare a class inside the class and define variable and methods.
class LwcPageLayout {
```

```
        // declare boolean variable
        flag =true;
        // declare array
        arrName =[];
        // declare json object
        jsonList ={};
        // declare variable with initialization
        x =10;
        // declare variable without initializtion
        prodName;


        // define custom methods
        display(){
                //code here
}

// define custom method
show(){
        // code here
}

// all name define inside the export which used any where with the help
of import keyword

export{
        name,
        listOfContact,
        clabel,
        LwcPageLayout,
        show,
}
```