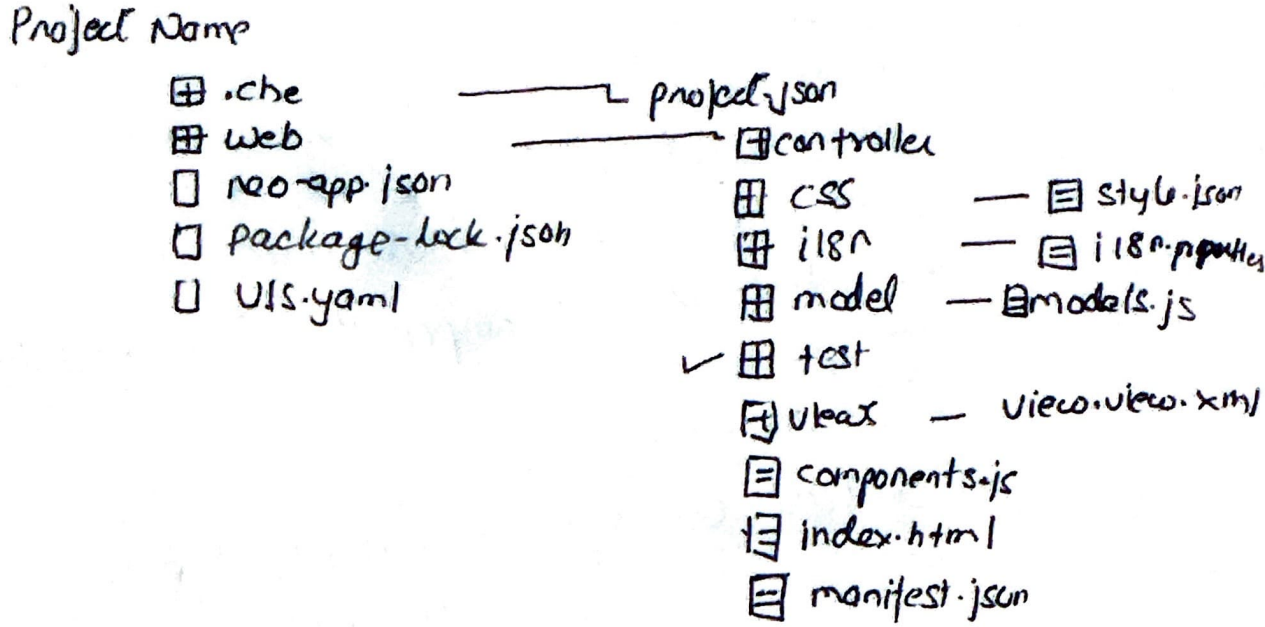
Structure of newly created project



→ When we run any application index.html is called

→ Launchpad configuration - how to load application using components.js using launchpad configuration

→ To run application directly using ~~launchpad~~ component.js we need to do this using launchpad configuration. we make file in launchpad configuration.

→

- After launch index.html, first select component.js file.

→ component file have path of manifest.json.

component.js

```
sap.ui.define(
    ["sap/ui/core/UIComponent",
     "sap/ui/Device",
     "demo/buttonDemo/model/models"
    ], function(UIComponent, Device, models)
    {
      "use strict";
    return UIComponent.extend("demo.buttonDemo.
                              component",
    {

      metadata: {
        manifest: "json"
      },

  /* component is initialized by UI5 automatically
  during the startup of the app and it calls
  the init method once.

  init: function() {
    UIComponent.prototype.init.apply(this, arguments);
    // enable routing
      this.getRouter().initialize();
  // set the device model
      this.setModel(models.createDeviceModel(),
                    "device");
    }
  });
});
```

The application runs with index.html. and we can't run it directly on component.js because we need to do launchpad configuration to run it directly from component.js.
In the launchpad we create tiles

First, we call index.html, which then calls the component.js, allowing us to determine which data will be loaded.
As we proceed, we see that it has path of manifest.json, and then the application goes into manifest.json.

Manifest.json :

```
{
 "version": "1.12.0",
 "sap.app": {
        "id": "MRL.dsm.DEMO",
        "type": "application",
        "i18n": "i18n/i18n.properties",
        "applicationVersion": {
            "version": "1.0.0"
        },
    "title": "{{appTitle}}",
    "description": "{{appDescription}}",
    "sourceTemplate": {
     "id": "servicecatalog.connectivityComponent
                ForManifest",
     "version": "0.0.0"
 },
```

```
"dataSources": {
        "ZMM_STK_303_SRV": {
        "uri": "/sap/opu/odata/sap/ZMM_STK_303_
                                        SRV/",
      "type": "OData",
      "settings": {
        "localUri": "localService/metadata.xml"
      }
```

When we create an application in S4 Hana, it creates two files, component.js and manifest.js. Component.js is the main file which is loaded after index.html and component.js calles manifest.json. It is recommended that component.js should have less code and all the data should be loaded in manifest.js because if we write more data in this file, it will increase the application loading time.

A vendor provided an OData service and asked to send the user information to know for which user this service call was made.

~~index.html calls mai~~

First we load index.html and it calls component.js and inside component.js there is manifest.json. Manifest.json have data source information within tag (key) datasource.

If we want to add a data service, we can do it in manifest.json editor.

Previously, old app development, eclipse version before WebID, all the code was in component. However, in SAP webide, it is considered ~~to keep~~ better to keep the code in two seperate files - component.js and manifest.json

When we now create a new application in WebID, it generates two files, component.js and manifest.json. The routing, oData and all other codes go to manifest.json and it is considered better to keep the component.js ~~file empty~~. as blank as possible because the initial load goes to the component file. If there is too much data, the application loading time will increase, affecting the application performance.

We add necessary data to the component, which is important for the initial launch of the application.

For example, if we need user information oData, we load in the component itself to get the initial oData for the user for further use.

For example if we need to display user related information on the application's first form, we need to load the initial user data. We need to send the user's information via oData.
After getting user data we can further use it at any part of app.

Within the application, we represent each form using View.xml. If our View form is S1.View.xml, it will create a corresponding S1.controller.js with an ID.
The handling of the data to be displayed in S1.View.xml is done within the S1.controller.js

If there are multiple controls in View.xml and different data needs to be added to them, the code for this will be in the controller.js. We will write it in the corresponding View's Controller.js file.

If my application's view has a form with a table view, a few combo boxes, and the data should be displayed as soon

the form loads, then we write the code in the view's controller.js file. We'll write seperate functions for each control, like table and combo boxes, to refresh and load the data, and we will call those functions in the onInit() method.

S1.controller.js

```
sap.ui.define([
    "sap/ui/core/mvc/Controller"
], function(Controller)
    {
        "use strict";
    return Controller.extend("MRL.dsm.controller.S1", {

        onInit : function() {
    this.loadTableData();
    this.loadComboData();
        },
        loadTableData : function() {
        loadComboData : function() {
        loadBasicformdata : function() {
        }
    });
});
```