ArcGIS API for JavaScript Fundamentals

Omar Elhadi

Installing ArcGIS API locally

1. Download ArcGIS API archive file
2. Extract arcgis_js_api to your localhost folder
3. Test the URL:

   http://localhost/arcgis_js_api/library/4.14/init.js
4. Open the **init.js** file which can be located in similar directory:

   C:\wamp\www\arcgis_js_api\library\4.14

   Or C:\ Inetpub\wwwroot\arcgis_js_api\library\4.14
5. Replace : https://[HOSTNAME_AND_PATH_TO_JSAPI]

   With http://localhost/arcgis_js_api/library/4.14/
6. Repeat the same step with the file **dojo.js** which can be located in similar directory:

   C:\wamp\www\arcgis_js_api\library\4.14\dojo

   Or C:\ Inetpub\wwwroot\arcgis_js_api\library\4.14\dojo
7. Now replace the URLs in the Local_API_Test file

```
<link rel="stylesheet" href="http://localhost/arcgis_js_api/library/4.14/esri/themes/light/main.css" />
<script src="http://localhost/arcgis_js_api/library/4.14/dojo/dojo.js"></script>
```

With:

```
http://localhost/arcgis_js_api/library/4.14/esri/themes/light/main.css
```

And

```
http://localhost/arcgis_js_api/library/4.14/dojo/dojo.js
```

1) My First Map

Add references to the CSS and API

```
<link rel="stylesheet" href="http://localhost/arcgis_js_api/library/4.14/esri/themes/light/main.css" />
<script src="http://localhost/arcgis_js_api/library/4.14/dojo/dojo.js"></script>
```

References to the online version

```
<linkrel="stylesheet" href="https://js.arcgis.com/4.14/esri/themes/light/main.css"/>
    <script src="https://js.arcgis.com/4.14/"></script>
```

Fetching the modules

```
 require(["esri/Map", "esri/views/MapView"], function(Map, MapView) {

});//======= Require ==============
```

Define the map

```
var map = new Map({
          basemap: "hybrid" // topo, osm, streets...
        });
```

Define the view

```
var view = new MapView({
          container: "viewDiv",
          map: map,
          zoom: 13,
          center: [58.5, 23.58], // Sets center point of view using longitude,latitude
        });
```

2) My First Scene

Add references to the CSS and API

```
<link rel="stylesheet" href="http://localhost/arcgis_js_api/library/4.14/esri/themes/light/main.css" />
<script src="http://localhost/arcgis_js_api/library/4.14/dojo/dojo.js"></script>
```

References to the online version

```
<linkrel="stylesheet" href="https://js.arcgis.com/4.14/esri/themes/light/main.css"/>
    <script src="https://js.arcgis.com/4.14/"></script>
```

Fetching the modules

```
 require(["esri/Map", "esri/views/SceneView"], function(Map, SceneView) {


});//======= Require ==============
```

Define the map

```
var map = new Map({
          basemap: "topo-vector", // streets-navigation-vector, streets-relief-vector...
      ground: "world-elevation"
        });
```

Define the view

```
var view = new SceneView({
      container: "viewDiv",
      map: map,
      camera: {
        position: {  // observation point
          x: 57.5,
          y: 23,
          z: 45000 // altitude in meters
        },
        tilt: 30  // perspective in degrees
      }
    });
```

3) Basemap Toggle

Import target modules and functions

```
"esri/widgets/BasemapToggle",
"esri/widgets/BasemapGallery"
```

```
BasemapToggle, BasemapGallery
```

Define basemap toggle

```
var basemapToggle = new BasemapToggle({
        view: view,
        nextBasemap: "satellite"
      });
```

Add basemap toggle to the view

```
view.ui.add(basemapToggle, "bottom-right");
```

Types of layers

Subclasses: BaseDynamicLayer , BaseElevationLayer , BaseTileLayer , BuildingSceneLayer , CSVLayer , ElevationLayer , FeatureLayer , GeoJSONLayer , GeoRSSLayer , GraphicsLayer , GroupLayer , ImageryLayer , IntegratedMeshLayer , KMLLayer , MapImageLayer , MapNotesLayer , PointCloudLayer , SceneLayer , TileLayer , UnknownLayer , UnsupportedLayer , VectorTileLayer , WMSLayer , WMTSLayer , WebTileLayer

4) Add CSVLayer

Import target modules and functions

```
"esri/layers/CSVLayer"
```

```
CSVLayer
```

Define CSV layer

```
var myLayer = new CSVLayer({
          url: "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_week.csv"
        });
```

Add layer to the map

```
map.add(myLayer);
```

5) Add MapImageLayer

Import target modules and functions

```
"esri/layers/MapImageLayer"
```

```
MapImageLayer
```

Define MapImageLayer layer

```
var myLayer = new MapImageLayer({
url: "https://nowcoast.noaa.gov/arcgis/rest/services/nowcoast/forecast_meteoceanhydro_sfc_ndfd_time/MapServer",
        //https://sampleserver6.arcgisonline.com/arcgis/rest/services/Census/MapServer
        //https://sampleserver6.arcgisonline.com/arcgis/rest/services/SampleWorldCities/MapServer

        });
```

Add layer to the map

```
map.add(myLayer);
```

6) Add FeatureLayer

Import target modules and functions

```
"esri/layers/FeatureLayer"
```

```
FeatureLayer
```

Define FeatureLayer layer

```
var myLayer = new FeatureLayer({
url: "https://services6.arcgis.com/nEMEkLg8rZV7Ijyb/ArcGIS/rest/services/SudanMap/FeatureServer/2" //0,1
//https://services.arcgis.com/DCPX1PuggGH4Tici/arcgis/rest/services/Sudan%20Project%20Locations%20Density/FeatureServer
      });
```

Add layer to the map

```
map.add(myLayer);
```

7) FeatureLayers order

Restore default order

```
map.add(points,0);
```

Import in correct order

```
map.addMany([polygons,lines,points]);
```

8) Labeling

Define label class

```
var labelClass =
        {
          symbol:
          {
            type: "text",
          },
          labelPlacement: "above-center",
          labelExpressionInfo:
          {
          expression: "$feature.mag”
          },

         }
```

Define labeling Info source

```
labelingInfo: [labelClass]
```

Symbols

ESRI symbols:

- CIMSymbol
- ExtrudeSymbol3DLayer
- FillSymbol3DLayer
- IconSymbol3DLayer
- LineSymbol3DLayer
- ObjectSymbol3DLayer
- PathSymbol3DLayer
- TextSymbol3DLayer
- WaterSymbol3DLayer
- LabelSymbol3D
- LineSymbol3D
- MeshSymbol3D
- PointSymbol3D
- PolygonSymbol3D
- Font
- PictureFillSymbol
- PictureMarkerSymbol
- SimpleFillSymbol
- SimpleLineSymbol
- SimpleMarkerSymbol
- TextSymbol
- WebStyleSymbol

Possible Values:`"simple-marker"`|`"picture-marker"`|`"simple-line"`|`"simple-fill"`|`"picture-fill"`|`"text"`|`"shield-label-symbol"`|`"point-3d"`|`"line-3d"`|`"polygon-3d"`|`"web-style"`|`"mesh-3d"`|`"label-3d"`|`"cim"`

Renderer

Subclasses: ClassBreaksRenderer , DictionaryRenderer , DotDensityRenderer , HeatmapRenderer , SimpleRenderer , UniqueValueRenderer

Since: ArcGIS API for JavaScript 4.0

Renderers define how to visually represent each feature in one of the following layer types:

- FeatureLayer
- SceneLayer
- MapImageLayer
- CSVLayer
- StreamLayer

There are several types of renderers available for visualizing data. Each serves a different purpose, allowing you to explore your data and tell a visual story about it by combining geography and statistics. Most cartographic visualizations fall into one of the following categories.

Visualization type	Renderer
Location only	SimpleRenderer, HeatmapRenderer
Unique (typed) values	UniqueValueRenderer
Class breaks	ClassBreaksRenderer
Continuous color/size	SimpleRenderer or UniqueValueRenderer with visualVariables
Multivariate	SimpleRenderer or UniqueValueRenderer with visualVariables

9) Points layer symbology

Define layer Renderer

```
var layerRenderer =
       {
         type: "simple",
         symbol:
         {
           type: "simple-marker",
           style: "cicle",
           color: "blue",
           size: "8px",
         }
       }
```

Define Renderer source

```
renderer: layerRenderer
```

10) Polylines layer symbology

Define layer Renderer

```
var layerRenderer  = {
        type: "simple",
        symbol: {
          type: "simple-line",
          color: "#BA55D3",
          width: "2px",
          style: "solid",
        },

      }
```

11) Polygons layer symbology

Define layer Renderer

```
var layerRenderer =
        {
          type: "simple",
          symbol:
          {
            type: "simple-fill",
            color: [ 255, 128, 0, 0.5 ],
            outline:
            {
              width: 1,
              color: "white"
            }
          },

        }
```

12) ClassBreaksRenderer

Define layer Renderer

```
Var layerRenderer =
        {
          type: "class-breaks",
          field: "Total_Pop",
          classBreakInfos:
          [
            {
              minValue: 11000,
              maxValue: 200000,
              symbol:
              {
                type: "simple-fill",
                color: "#995874",
                outline:
                {
                  width: 1,
                  color: "white"
                }
              },
            },

            {
              minValue: 200001,
              maxValue: 400000,
                symbol:
              {
                type: "simple-fill",
                color: "#993560",
                outline:
                {
                  width: 1,
                  color: "white"
                }
              },
            },

            {
              minValue: 400001,
              maxValue: 700000,
              symbol:
              {
                type: "simple-fill",
                color: "#991E53",
                outline:
                {
                  width: 1,
                  color: "white"
                }
              },
            },

            {
              minValue: 700001,
              maxValue: 10000000,
              symbol:
              {
                type: "simple-fill",
                color: "#990344",
                outline:
                {
                  width: 1,
                  color: "white"
                }
              },
            },
          ]
        }
```

13) DotDensityRenderer

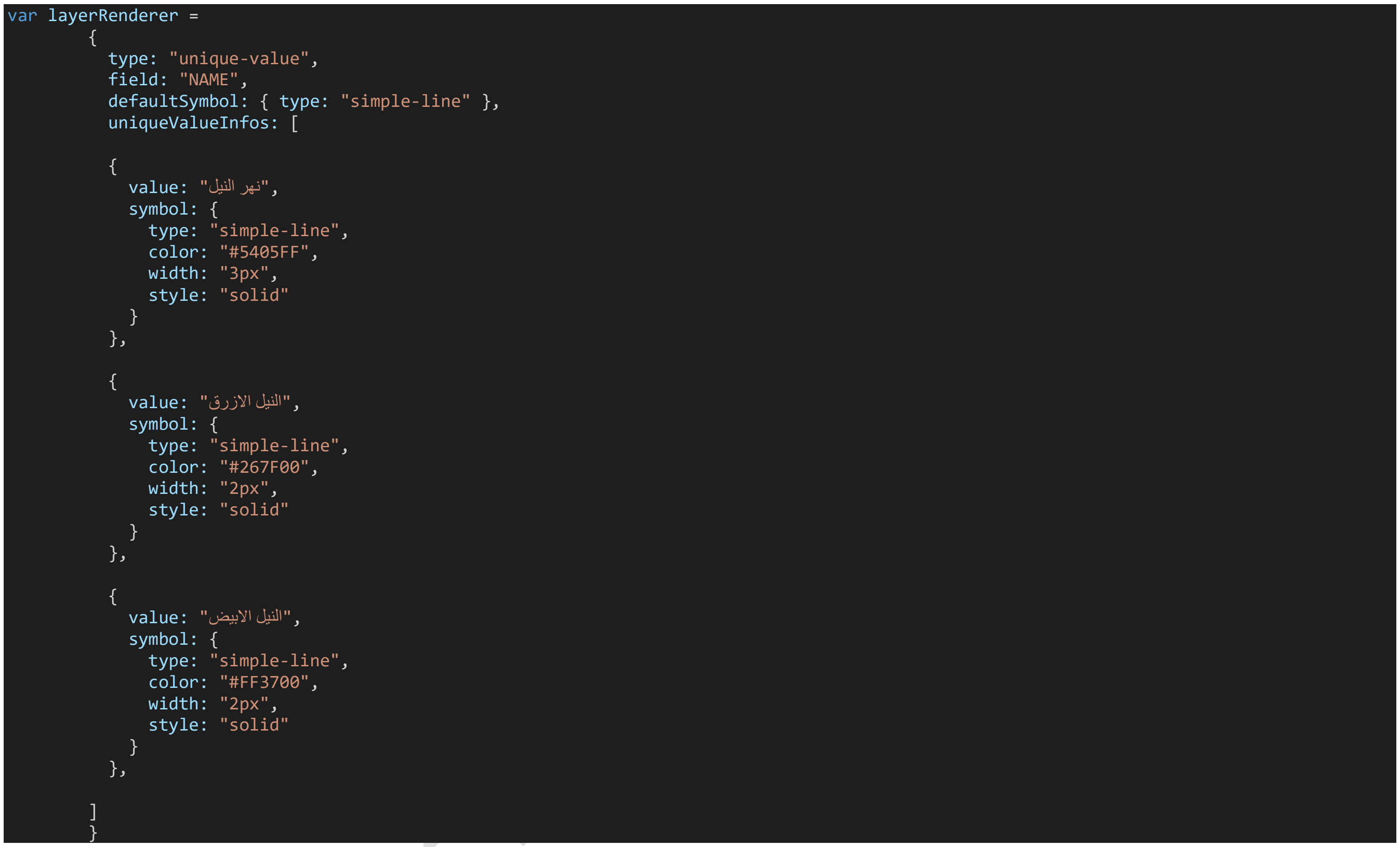
Define layer Renderer

```
var layerRenderer =
        {
          type: "dot-density",
          dotValue: 500,
          dotSize : 2,
          referenceScale: 100000,
          outline :
          {
            color: [ 150,150,150, 0.2 ],
            width: 0.5
          },
          attributes:
          [
            {
              field: "Total_Pop",
              label: "Male",
              color: "yellow"
            },

          ]
        }
```

14) UniqueValueRenderer [Line]

Define layer Renderer



```
var layerRenderer =
        {
          type: "unique-value",
          field: "NAME",
          defaultSymbol: { type: "simple-line" },
          uniqueValueInfos: [

          {
            value: "نهر النيل",
            symbol: {
              type: "simple-line",
              color: "#5405FF",
              width: "3px",
              style: "solid"
            }
          },

          {
            value: "النيل الازرق",
            symbol: {
              type: "simple-line",
              color: "#267F00",
              width: "2px",
              style: "solid"
            }
          },

          {
            value: "النيل الابيض",
            symbol: {
              type: "simple-line",
              color: "#FF3700",
              width: "2px",
              style: "solid"
            }
          },

        ]
        }
```

15) UniqueValueRenderer [Polygon]

Define layer Renderer

```
var layerRenderer =
        {
          type: "unique-value",
          field: "Loc_Eng",
          defaultSymbol: { type: "simple-fill", color: "rgba(63, 40, 102, 0.3)" },
          uniqueValueInfos: [

            {
            value: "El Malha",
            symbol: {
              type: "simple-fill",
              color: "blue"
            }
          },

          {
            value: "Haya",
            symbol: {
              type: "simple-fill",
              color: "green"
            }
          },

          {
            value: "Buram",
            symbol: {
              type: "simple-fill",
              color: "red"
            }
          }

        ]
        }
```

16) HeatmapRenderer

Define layer Renderer

```
var layerRenderer =
        {
          type: "heatmap",
          colorStops: [
            { color: "rgba(63, 40, 102, 0)", ratio: 0 },
            { color: "#4e2d87", ratio: 0.2},
            { color: "#563098", ratio: 0.4},
            { color: "#5d32a8", ratio: 0.6 },
            { color: "#6735be", ratio: 0.8 },
            { color: "#ffff00", ratio: 1 }
          ],
          maxPixelIntensity: 500,
          minPixelIntensity: 0
        }
```

17) Point Graphic

Import target modules and functions

```
"esri/Graphic"
```

```
Graphic
```

Create a point geometry

```
var point = {
          type: "point",
          longitude: 0,
          latitude: 0
        };
```

Create a symbol for drawing the point

```
var markerSymbol = {
          type: "picture-marker",
            url: "https://i.imgur.com/n9ZE9Hn.png",
            width: "45px",
            height: "45px"
        };
```

Create a graphic and add the geometry and symbol to it

```
 var pointGraphic = new Graphic({
          geometry: point,
          symbol: markerSymbol
        });
```

Add Graphic to view

```
view.graphics.add(pointGraphic);
```

18) Line Graphic

Create a line geometry

```
var polyline = {
          type: "polyline",
          paths: [
            [0, 0],
            [0, 25],
            [20, 25],
            [20, 0],
            [0, 0],
          ]
        }
```

Create a symbol for drawing the line

```
var lineSymbol = {
          type: "simple-line",
          color: "#F2D11D",
          width: 4
        }
```

Create a graphic and add the geometry and symbol to it

```
var polylineGraphic = new Graphic({
          geometry: polyline,
          symbol: lineSymbol,
        });
```

Add Graphic to view

```
view.graphics.add(polylineGraphic);
```

19) Polygon Graphic

Create a Polygon geometry

```
var polygon = {
          type: "polygon",
            rings: [
            [0, 0],
            [0, 25],
            [20, 25],
            [20, 0],
            [0, 0],
            ]
          }
```

Create a symbol for rendering the polygon

```
var fillSymbol = {
          type: "simple-fill",
          color: [115, 5, 235, 0.5],
          outline: {
            color: "#F2D11D",
            width: 3
          }
        }
```

Create a graphic and add the geometry and symbol to it

```
var polygonGraphic = new Graphic({
          geometry: polygon,
          symbol: fillSymbol
        });
```

Add Graphic to view

```
view.graphics.add(polygonGraphic);
```

20) Graphic PopupTemplate

Create attributes for the graphic

```
var graphicAttr = {
          Name: "Smiley",
          Mood: "Happy",
          Reason: "https://www.youtube.com/watch?v=hy1I25JFjX0"
        }
```

Define the graphic attributes and popup template

```
attributes: graphicAttr ,
popupTemplate: {
            title: "{Name}",
            content: [
              {
                type: "fields",
                fieldInfos: [
                  {
                    fieldName: "Name"
                  },
                  {
                    fieldName: "Mood"
                  },
                  {
                    fieldName: "Reason"
                  }
                ]
              }
            ]
          }
```

Try this: https://i.imgur.com/ynlVmhR.png

21) FeatureLayer Popup

Create a popup template

```
var myPopupTemplate = {
            title: "{Loc_Eng}",
            content: [
              {
                type: "fields",
                fieldInfos: [
                  {
                    fieldName: "Total_Pop",
                   // label: "Population"
                  },
                  {
                    fieldName: "Total_M",
                    label: "Male",
                  },
                  {
                    fieldName: "Total_Fe",
                    label: "Female",
                  },

                ]
              }
            ]
          }
```

Refer to the popup template

```
popupTemplate: myPopupTemplate
```

22) Legend

Import target modules and functions

```
"esri/widgets/Legend"
```

```
Legend
```

Create a legend

```
var legend = new Legend({
            view: view,
            layerInfos: [
              {
                layer: polygons,
              },
            ],
          });
```

Add legend to the view

```
view.ui.add(legend, "bottom-left");
```

23) Layerlist

Import target modules and functions

```
"esri/widgets/LayerList"
```

```
LayerList
```

Create a Layerlist

```
var layerList = new LayerList({
   view: view
});
```

Add Layerlist to the view

```
view.ui.add(layerList, "top-right");
```

24) Widgets

Compass

```
"esri/widgets/Compass"
Compass
```

```
var compass = new Compass({
          view: view
        });
        view.ui.add(compass, "top-left");
```

Fullscreen

```
"esri/widgets/Fullscreen"
Fullscreen
```

```
fullscreen = new Fullscreen({
          view: view
        });
        view.ui.add(fullscreen, "top-left");
```

Home

```
"esri/widgets/Home"
Home
```

```
var homeWidget = new Home({
          view: view
        });
        view.ui.add(homeWidget, "top-left");
```

Search

```
"esri/widgets/Search"
Search

var searchWidget = new Search({
            view: view
          });
          view.ui.add(searchWidget, {
            position: "top-right",
          });
```

ScaleBar

```
"esri/widgets/ScaleBar"
ScaleBar
```

```
var scaleBar = new ScaleBar({
          view: view,
        });
        view.ui.add(scaleBar, {
          position: "top-left"
        });
```

Sketch

```
"esri/widgets/Sketch",
      "esri/layers/GraphicsLayer"
Sketch,GraphicsLayer
```

First create a graphic layer

```
const myGraphicLayer = new GraphicsLayer();
```

Refer to the graphic layer within the map

```
layers: [myGraphicLayer]
```

Now create the sketch

```
const sketch = new Sketch({
          layer: myGraphicLayer,
          view: view,
        });
        view.ui.add(sketch, "top-right");
```

25) MapView Events – Click

Listen to clicks on the view

```
view.on("click", function(event){

            console.log("Clicked");

          });//===== View watch ============
```

26) MapView Events - Keyboard keys

Listen to the keyboard's keys

```
view.on("key-down", function(evt){
          console.log(evt);
          });
```

27) Filters

Create expressions array

```
var sqlExpressions = ["Total_Pop > 300000", "Total_Pop < 300000", "Loc_Eng = 'El Fashir'"];
```

Create select element

```
 var selectFilter = document.createElement("select");
          selectFilter.setAttribute("class", "esri-widget esri-select");
          selectFilter.setAttribute("style", "width: 275px; font-family: Avenir Next W00; font-
size: 1em;");
```

Define the select element options

```
          sqlExpressions.forEach(function(sql){
            var option = document.createElement("option");
            option.value = sql;
            option.innerHTML = sql;
            selectFilter.appendChild(option);
          });
```

Add select element to the view

```
view.ui.add(selectFilter, "top-right");
```

Listen to select element

```
selectFilter.addEventListener('change', function (event) {
            setFeatureLayerFilter(event.target.value);
          });
```

Fire a function when select element's value changes

```
function setFeatureLayerFilter(expression) {
            featureLayer.definitionExpression = expression;
          }
```

28) Customized Filter

Add div element to the view

```
view.ui.add("queryDiv", "top-right");
```

Listen to filter firing button to unleash filtering function

```
document.getElementById("filter").onclick = filterFun;
```

Define filtering function

```
          function filterFun() {

            var field = document.getElementById("queryField").value;

            var sign = document.getElementById("querySign").value;

            var population = document.getElementById("queryValue").value;

            var filterExp = field+sign+population;

            featureLayer.definitionExpression = filterExp;

          }
```

Stop filtering

```
document.getElementById("stopFilter").onclick = stopFilterFun;
          function stopFilterFun() {
            var filterExp = "";
            featureLayer.definitionExpression = filterExp;
          }
```

29) Spatial Query

Start listening to the view

```
view.on("click", function(event){

});//===== View watch ===========
```

Create a query for the layer

```
var query = featureLayer.createQuery();
            query.geometry = view.toMap(event);  // the point location of the pointer
            query.distance = 1;
            query.units = "miles";
            query.spatialRelationship = "intersects";
            query.returnGeometry = true;
```

Run the query

```
featureLayer.queryFeatures(query)
              .then(function(response){


});
```

Extract geometries from the response

```
var featuresGeometries = response.features.map(function(feature) {
                  return feature.geometry;
                });
```

Take one geometry

```
var resultGeometry = featuresGeometries[0];
```

Define a graphic and add it to the view

```
var fillSymbol = {
                 type: "simple-fill",
                 color: [50, 50, 50, 0.5],

               }
               var polygonGraphic = new Graphic({
                 geometry: resultGeometry,
                 symbol: fillSymbol
               });
               view.graphics.add(polygonGraphic);
```

Remove all graphics when starting new session

```
view.graphics.removeAll();
```

30) Final Project

Watch the view for key-down event

```
view.on("key-down", function(evt){
          if (evt.key === "m"){
            var menuStatus = document.getElementById("menu").style;
            if(menuStatus.display === "none")
              menuStatus.display = "block";
              else
                menuStatus.display = "none";
          }
        });//========== view watch =============
```

Create a legend

```
var legend = new Legend({
                view: view,
                layerInfos: [
                  {
                    layer: polygons,
                    title: "LOCALITIES"
                  },
                  {
                    layer: points,
                    title: "SETTLEMENTS"
                  }]
              });
```

Link legend with the check box

```
const legendCheck = document.getElementById('legendCheck');
          legendCheck.addEventListener('change', (event) => {
            if (event.target.checked) {
              view.ui.add(legend, "bottom-left");
            } else {
              view.ui.remove(legend);
            }
          });
```

Create a layerlist

```
var layerList = new LayerList({
              view: view
            });
```

Link layerlist with the check box

```
const layerListCheck = document.getElementById('layerListCheck');
          layerListCheck.addEventListener('change', (event) => {
            if (event.target.checked) {

            view.ui.add(layerList, "bottom-left");
            } else {
              view.ui.remove(layerList);
            }
          });
```

Create add layer tool

```
document.getElementById("addLayer").onclick = addLayerFun;
          function addLayerFun() {
            var featureLayer = new FeatureLayer({
            url: document.getElementById("layerURL").value
          });
          map.add(featureLayer);
          }
```

Create style changer tool

```
document.getElementById("changeStyle").onclick = changeStyleFun;
          function changeStyleFun() {
            var firstColor = document.getElementById('color1').value;
            var secondColor = document.getElementById('color2').value;
            layerRenderer.visualVariables = [{

              }];

            }
```

Spatial Query

Define a graphic layer for querying

```
const queryLayer = new GraphicsLayer();
```

Define the sketch

```
const sketch = new Sketch({
            layer: queryLayer,
            view: view,
            availableCreateTools: ["polygon"],
            creationMode: "single",
          });
```

Set the first action for query function trigger

```
document.getElementById("startQuery").onclick = startQueryFun;
        function startQueryFun() {
          map.add(queryLayer);
          view.ui.add(sketch, "top-left");

          sketch.on("create", function(event) {
            if (event.state === "complete") {
              queryFun(event.graphic.geometry);
            }
          });
          }//=========startQueryFun ==================
```

Main query function

```
function queryFun(geom) {
            var query = polygons.createQuery();
            query.geometry = geom;
            query.spatialRelationship = "intersects";
            query.returnGeometry = true;

            polygons.queryFeatures(query)
              .then(function(response){
                var geometriesArray = response.features.map(function(feature) {
                  return feature.geometry;
                });

                const fillSymbol = {
                    type: "simple-fill",
                    color: [250, 250, 250],

                  };
                geometriesArray.forEach(drawResultFun);
                function drawResultFun(geom)
                {
```

```
            var polygonGraphic = new Graphic({
              geometry: geom,
              symbol: fillSymbol
            });
            view.graphics.add(polygonGraphic);
          }
        });
    }//========= queryFun ==================
```