ArcGIS JavaScript API
Demonstration Documentation

By Colin Haroutunian & Neal Sabin

I. Objectives

A. By the end of this documentation, you should be able to:

1. Download the necessary environment to start developing web maps using the ArcGIS JavaScript API.
2. Integrate feature layers into your web map.
3. Integrate widgets into your web map.
4. Create graphics layers in your web map.
5. Add styling to the features and graphic layers.

II. Download Starter Files

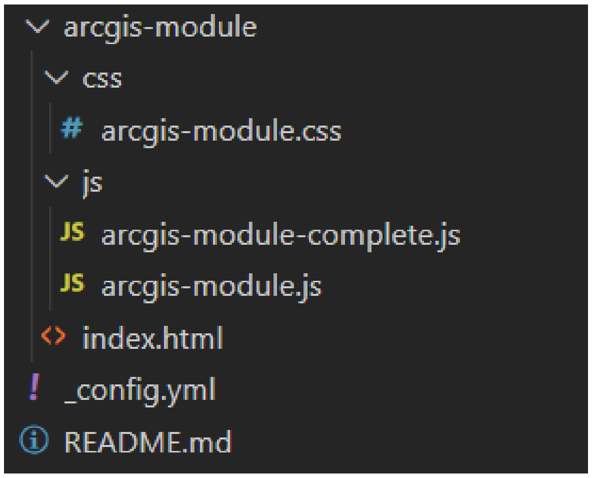
A. Follow the link below to go to the project repository. Select 'Download ZIP' to download the repository.

GitHub repo: https://github.com/coharou/arcgis-module

B. Once downloaded, extract the files to where you would like the folder to reside.

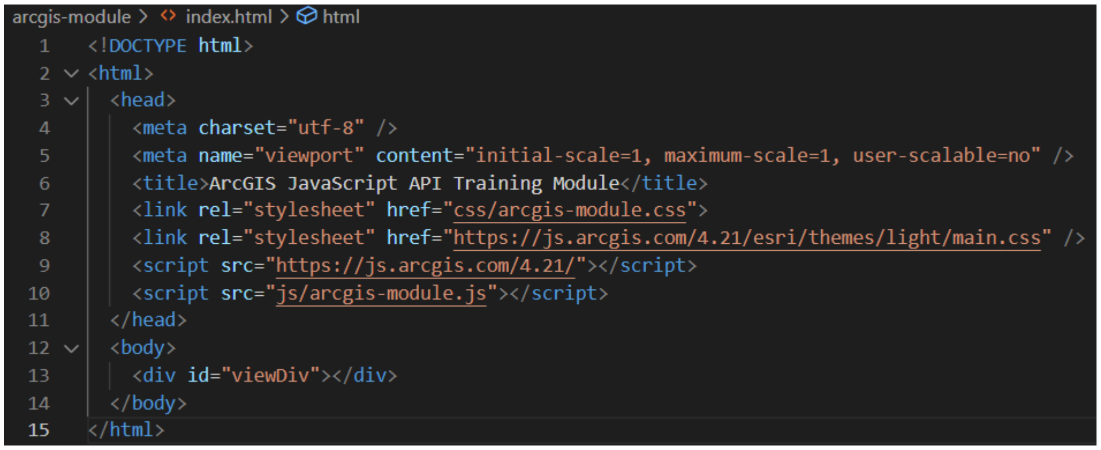
C. Open Visual Studio Code or your preferred code editor.

1. Click 'File' > 'Open Folder…' and navigate to the project folder you just downloaded.
2. With the 'css' and 'js' folder expanded, your folder structure should look like the screenshot below.



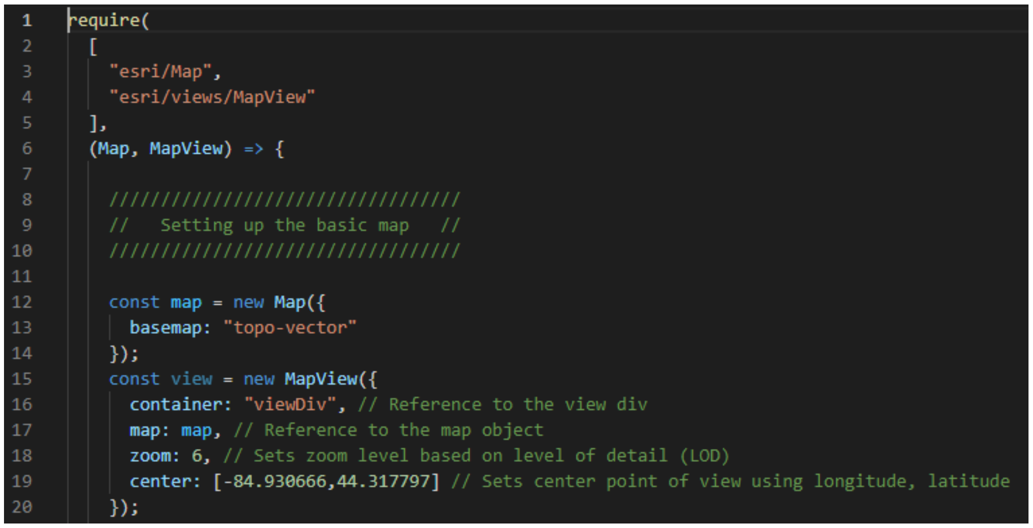
```
arcgis-module
  css
    arcgis-module.css
  js
    arcgis-module-complete.js
    arcgis-module.js
  index.html
_config.yml
README.md
```

3. Double click the ‘index.html’ file to open it. Your file should look like the screenshot below. This is a simple HTML template that references the project style sheet, the associated style reference to the ArcGIS JavaScript API, the actual ArcGIS JavaScript API, and the project JavaScript file. The body contains a div tag, with an id of “viewDiv”, that will be referenced in the project JavaScript file.



```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no" />
    <title>ArcGIS JavaScript API Training Module</title>
    <link rel="stylesheet" href="css/arcgis-module.css">
    <link rel="stylesheet" href="https://js.arcgis.com/4.21/esri/themes/light/main.css" />
    <script src="https://js.arcgis.com/4.21/"></script>
    <script src="js/arcgis-module.js"></script>
  </head>
  <body>
    <div id="viewDiv"></div>
  </body>
</html>
```

4. Double click on the ‘arcgis-module.js’ file to open it. Your file should look like the screenshot below.

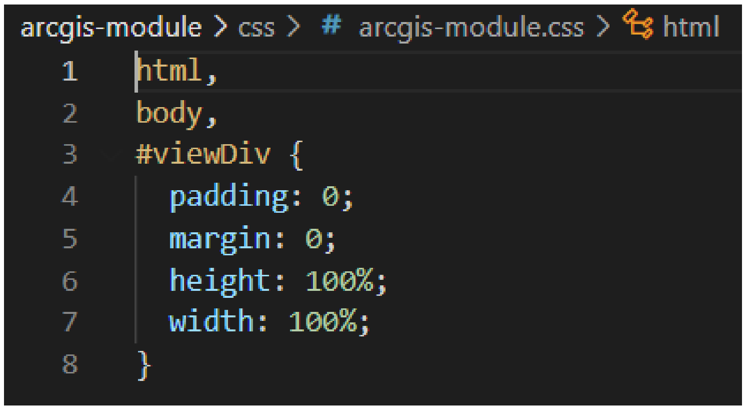


```javascript
1   require(
2     [
3       "esri/Map",
4       "esri/views/MapView"
5     ],
6     (Map, MapView) => {
7
8       /////////////////////////////////
9       //   Setting up the basic map   //
10      /////////////////////////////////
11
12      const map = new Map({
13        basemap: "topo-vector"
14      });
15      const view = new MapView({
16        container: "viewDiv", // Reference to the view div
17        map: map, // Reference to the map object
18        zoom: 6, // Sets zoom level based on level of detail (LOD)
19        center: [-84.930666,44.317797] // Sets center point of view using longitude, latitude
20      });
```

- In the code provided, we are first loading in two different modules, Map and MapView. These 2 modules are required for every map. Lines 12-14 create a new map object, which references the Map class. We specify one property, ‘basemap’, which sets the basemap to ‘topo-vector’.

- Lines 15-20 creates a new view object, which references the MapView class. Views reference nodes that serve as containers in HTML files.
    - For the ‘container’ property, we’re referencing the ‘viewDiv’ we set in our html file.
    - The ‘map’ property references the map object described earlier.
    - The ‘zoom’ property sets the zoom level of the web map.
    - The ‘center’ property specifies the center of the map as it is displayed in your browser.

5. Double click on the ‘arcgis-module.css’ file to open it. Your file should look like the screenshot below.


arcgis-module > css > # arcgis-module.css > html

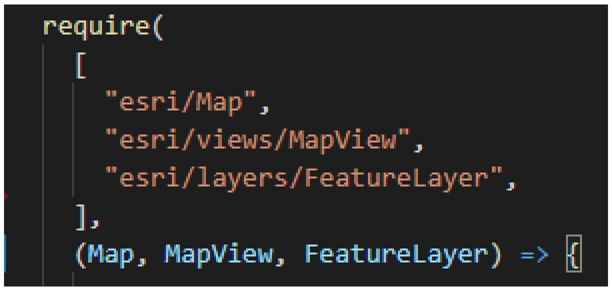
```css
html,
body,
#viewDiv {
  padding: 0;
  margin: 0;
  height: 100%;
  width: 100%;
}
```

This CSS styling ensures that the map fills the entire browser window.

III. Integrate Feature Layers

A. The FeatureLayer adds in feature services provided by ArcGIS.
   1. Feature services are special layers or effects that are not provided by the API at its basic level.
      a) In our case, we will be adding in a service that displays each county of the United States.

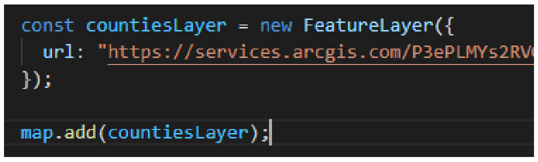
B. First, we are going to need to load in our FeatureLayer module.
   1. Return to the top of the script. Update the required module list and the function parameters to match the code below.



```
require(
  [
    "esri/Map",
    "esri/views/MapView",
    "esri/layers/FeatureLayer",
  ],
  (Map, MapView, FeatureLayer) => {
```

2. This will allow us to use the functionality provided by the FeatureLayer, as well as access the feature services.

C. We now need to add in our feature service, which comes with a URL.
   1. These services are available on the ArcGIS documentation website.
   2. This should be added beneath the "Adding features to the map" comment block of the script.
   3. Here is the URL that we will be using for the service: "https://services.arcgis.com/P3ePLMYs2RVChkJx/arcgis/rest/services/USA_Counties_Generalized/FeatureServer/0"
   4. Copy the code below, and type it beneath the "Adding features to the map" section of the file. You will want to add the URL from above into the url property of the layer.
      a) NOTE: part of the URL has been snipped out of the image.



```
const countiesLayer = new FeatureLayer({
  url: "https://services.arcgis.com/P3ePLMYs2RV
});

map.add(countiesLayer);
```

      b) The FeatureLayer object has now been created. It is a layer object, which is an extra "lens" that overlays the map.
         (1) The main property that the layer has is the URL, which allows the script to access the feature service from the ArcGIS services website.
      c) Beneath the FeatureLayer object, there is code that adds the layer to the map.
         (1) This code is required in order to implement the feature service to the map. Otherwise, it will not appear.

D. Open the HTML file in your browser, and make sure that there aren't any errors.
   1. You should see the entire United States filled in with orange counties.

IV. Integrate Widgets

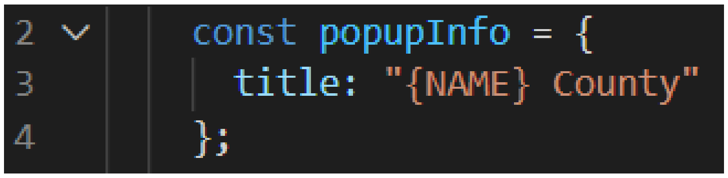
A. Once you have added the feature layer, it is now time to add some widgets. Widgets are reusable user-interface components and are key to providing a rich user experience. There are many widgets available in the ArcGIS JavaScript API, in this demonstration we will be working with the popup and layer list widgets.

B. Popups provide an easy way for users to access data from layers and graphics.

1. Open up the ‘arcgis-module.js’ file.

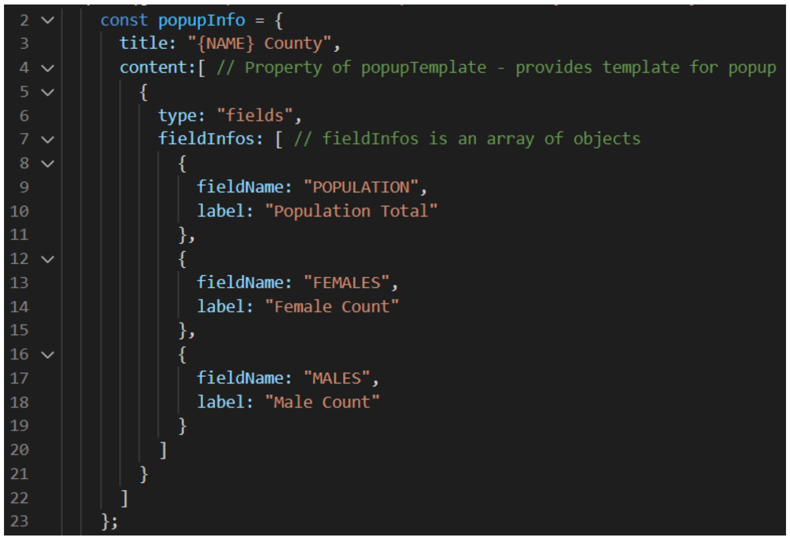
a) Above the FeatureLayer object you just added, create a PopupTemplate object and add a title. The value of the ‘title’ property will display at the top of the popup. Attributes within the feature layer are referenced by using the syntax {AttributeName}. Your code should look exactly like the screenshot below.

(1) NOTE: If this code is not added above the FeatureLayer object, the widget won't work!



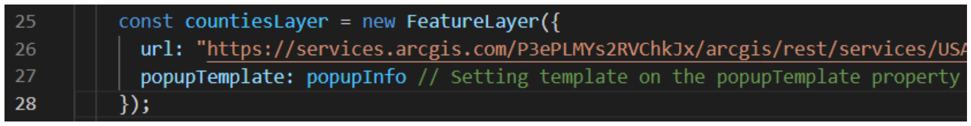
```javascript
const popupInfo = {
  title: "{NAME} County"
};
```

b) Add content to the Popup template. The content property provides a template defining the content of the popup for the feature layer. The popup information is derived from ‘fieldInfos’, which is an array of objects. Add the code as seen in the screenshot below to the popup template.



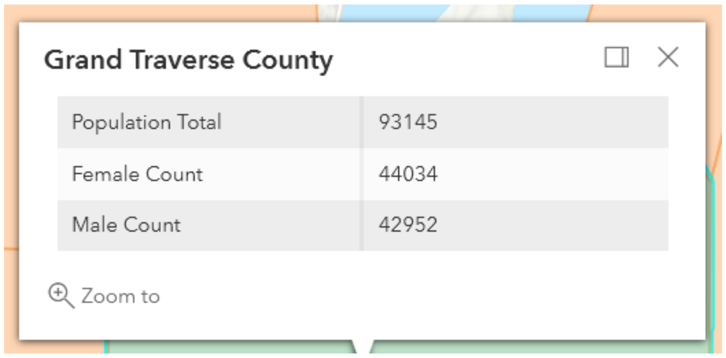
```javascript
const popupInfo = {
  title: "{NAME} County",
  content:[ // Property of popupTemplate - provides template for popup
    {
      type: "fields",
      fieldInfos: [ // fieldInfos is an array of objects
        {
          fieldName: "POPULATION",
          label: "Population Total"
        },
        {
          fieldName: "FEMALES",
          label: "Female Count"
        },
        {
          fieldName: "MALES",
          label: "Male Count"
        }
      ]
    }
  ]
};
```

c) Return to the object you created from the FeatureLayer class. You will need to add the ‘popupTemplate’ property and set it to the ‘popupInfo’ variable you just created. Your code should look like the screenshot below.



```
const countiesLayer = new FeatureLayer({
  url: "https://services.arcgis.com/P3ePLMYs2RVChkJx/arcgis/rest/services/USA
  popupTemplate: popupInfo // Setting template on the popupTemplate property
});
```

d) Now, open the map and click on a county of your choice. Make sure the popup that appears looks like the screenshot below.



Grand Traverse County

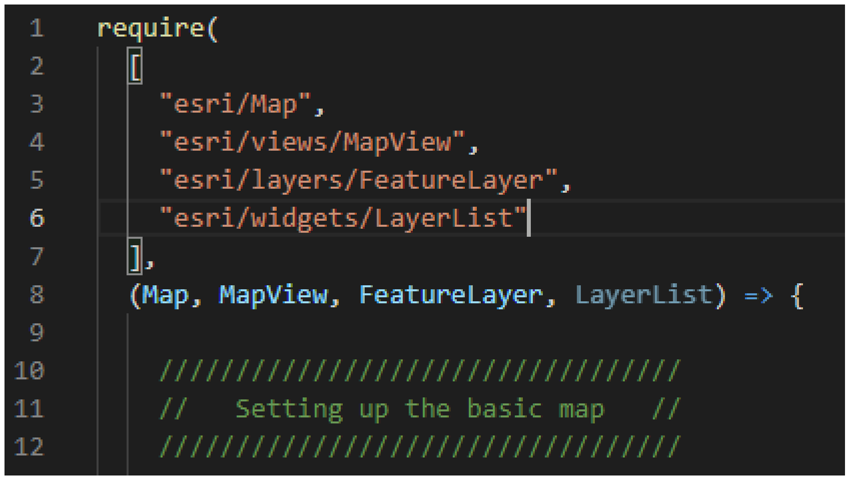
Population Total	93145
Female Count	44034
Male Count	42952

e) You are now done adding the popup widget.

C. The LayerList widget provides a way of displaying the layers that are being used in the map and the ability to turn them on or off.

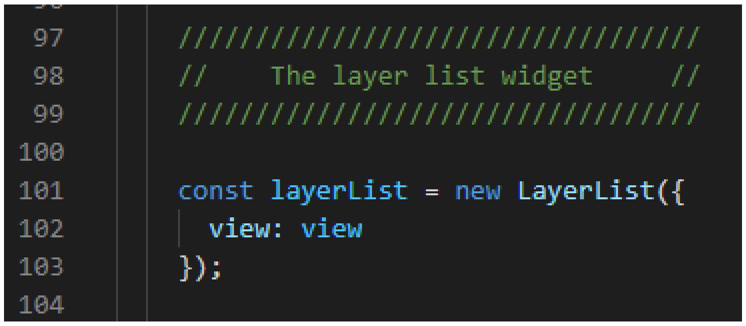
1. Keep the ‘arcgis-module.js’ file open.

a) Add the ‘LayerList’ module to the required statement, as shown below.



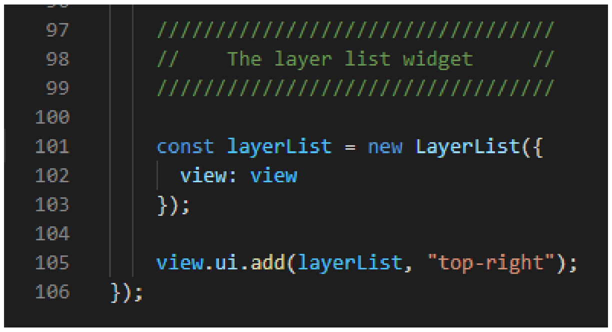
```
require(
  [
    "esri/Map",
    "esri/views/MapView",
    "esri/layers/FeatureLayer",
    "esri/widgets/LayerList"
  ],
  (Map, MapView, FeatureLayer, LayerList) => {

    ///////////////////////////////////
    //   Setting up the basic map   //
    ///////////////////////////////////
```

b) Scroll down to the bottom of the script, and underneath the comment block for "The layer list widget", create a new object called ‘layerList’, that is derived from the ‘LayerList’ class.
   (1) Set the ‘view’ property to the ‘view’ object mentioned earlier.
   (2) Your code should look like the screenshot below.



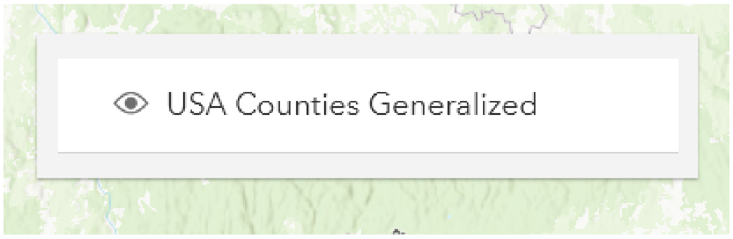
```
//////////////////////////////////
//    The layer list widget     //
//////////////////////////////////

const layerList = new LayerList({
  view: view
});

```

c) To add the widget to the map use the add method. Specify the component you are adding and the position of it. Your code should look like the screenshot below.



```
    //////////////////////////////////
    //    The layer list widget     //
    //////////////////////////////////

    const layerList = new LayerList({
      view: view
    });

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

d) Open the map, you should now see the layer list widget displaying the feature layer name. The widget should look like the screenshot below.



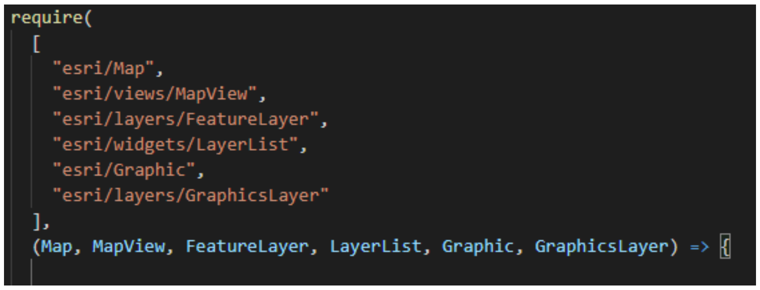
e) You are now done adding the layer list widget.

V. Create Graphic Layers

A. Now that you have added in the layer list widget, we need to add in another layer. This time, it will be for the GraphicsLayer.

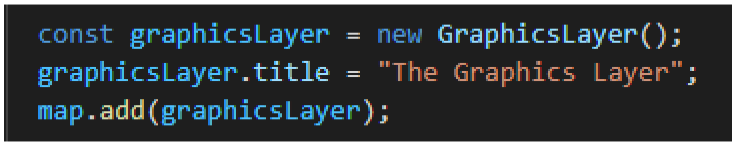
B. The GraphicsLayer stores the Graphic objects, which are drawings.
   1. Types of Graphic objects include:
      a) Point marks
      b) Icons or symbols
      c) Polylines
      d) Polygons
   2. These Graphic objects are overlaid onto the map. This means that you can add drawings to mark regions.

C. Since we're using both the GraphicsLayer and Graphic objects, we need to make sure to add these into our required modules.
   1. Update your modules and function to match the code below.



```
require(
  [
    "esri/Map",
    "esri/views/MapView",
    "esri/layers/FeatureLayer",
    "esri/widgets/LayerList",
    "esri/Graphic",
    "esri/layers/GraphicsLayer"
  ],
  (Map, MapView, FeatureLayer, LayerList, Graphic, GraphicsLayer) => {
```

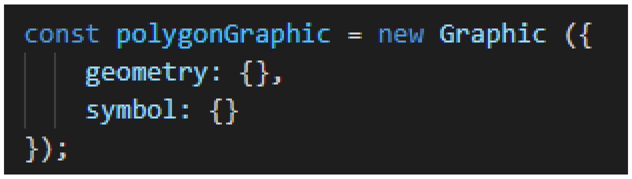
D. With the modules added in, let's get started by adding in the GraphicsLayer itself. You'll want to copy the code snippet below. You should type this beneath the "Adding graphics to the map" comment block.



```
const graphicsLayer = new GraphicsLayer();
graphicsLayer.title = "The Graphics Layer";
map.add(graphicsLayer);
```

   1. This code snippet creates the graphicsLayer object, adds a title to it, and then adds the layer to the map.
      a) This title will show up in the layer list widget that we added in earlier.

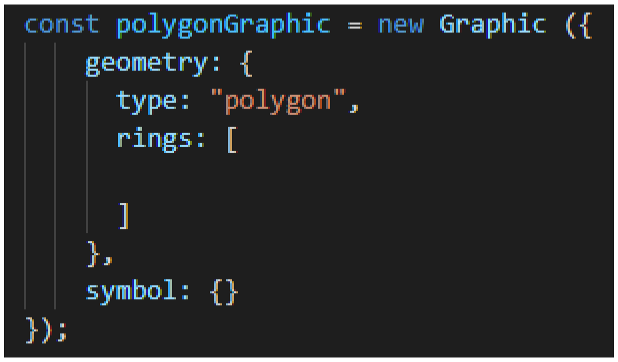
E. Now we need to add in the Graphic object to store our shape. You'll want to copy the code snippet below, and type this beneath the GraphicsLayer you just added.



```
const polygonGraphic = new Graphic ({
    geometry: {},
    symbol: {}
});
```

1. The geometry object will be storing information about our shape. In our case, we will be making a polygon that covers the state of Wyoming.

F. Since we're making a polygon, we need to include two properties in the geometry object: type and rings.
   1. The type property stores the type of shape that we'll be making. In this case, it will be "polygon".
   2. The rings object stores a list of coordinate objects.
      a) The easiest way to get these coordinates is by accessing Google Maps, right-clicking a location, and left-click the coordinates. These will be copied to your clipboard. When you paste these into the rings object, you'll want to swap the coordinates around, otherwise the shape won't be where you might expect it.
   3. Update the Graphic object to match the one below:

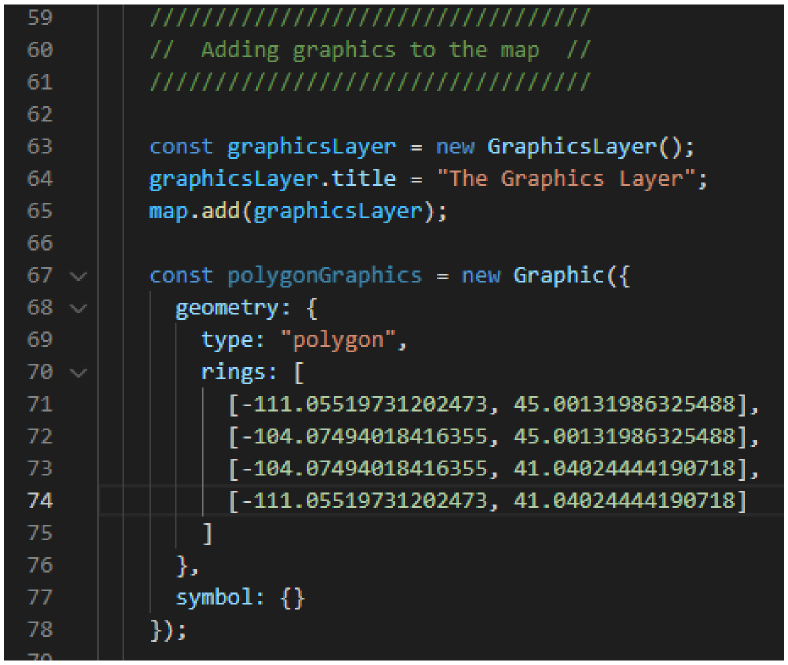


```
const polygonGraphic = new Graphic ({
    geometry: {
      type: "polygon",
      rings: [

      ]
    },
    symbol: {}
});
```

   4. Now, copy and paste the arrays below in the brackets for the rings.
      a) Make sure to not replace the brackets that are already added in!

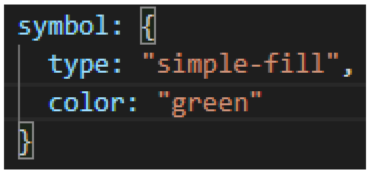
```
[-111.05519731202473, 45.00131986325488],
[-104.07494018416355, 45.00131986325488],
[-104.07494018416355, 41.04024444190718],
[-111.05519731202473, 41.04024444190718]
```

      b) Your code should now appear similar to the image below.



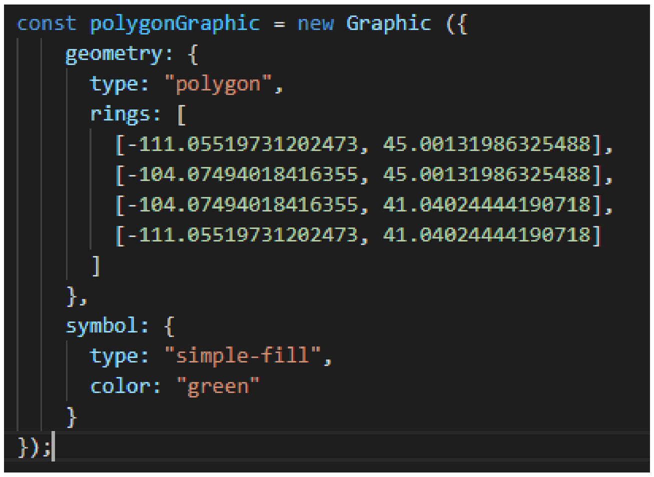
```
//////////////////////////////////
//  Adding graphics to the map  //
//////////////////////////////////

const graphicsLayer = new GraphicsLayer();
graphicsLayer.title = "The Graphics Layer";
map.add(graphicsLayer);

const polygonGraphics = new Graphic({
  geometry: {
    type: "polygon",
    rings: [
      [-111.05519731202473, 45.00131986325488],
      [-104.07494018416355, 45.00131986325488],
      [-104.07494018416355, 41.04024444190718],
      [-111.05519731202473, 41.04024444190718]
    ]
  },
  symbol: {}
});
```

G. Let's move onto the symbol object in our Graphic object. It's going to be simple for now, but we'll return to it later to add in new styling.
   1. Every symbol object needs a type property. This describes how the shape will be filled in.
      a) For instance, the "simple-fill" that we will be using fills in the shape with one color. In this case, the color will be green.
   2. Aside from the symbol, we will also be adding in a simple color to the shape. Copy the code below into the symbol property.



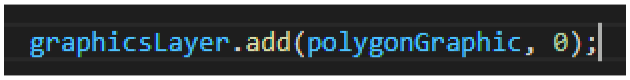
```
symbol: {
  type: "simple-fill",
  color: "green"
}
```

H. Having added in the geometry and symbol objects, your polygon object should now look like the image below.



```
const polygonGraphic = new Graphic ({
    geometry: {
      type: "polygon",
      rings: [
        [-111.05519731202473, 45.00131986325488],
        [-104.07494018416355, 45.00131986325488],
        [-104.07494018416355, 41.04024444190718],
        [-111.05519731202473, 41.04024444190718]
      ]
    },
    symbol: {
      type: "simple-fill",
      color: "green"
    }
});
```

I. Now that we've finished adding in the polygon Graphic object, we need to add it to our layer so that it appears on the map.
   1. Similar to how you would add a FeatureLayer to a map, we're going to be adding the Graphic object to the GraphicsLayer object.
   2. Copy the code below, and type it beneath the new polygon object.



```
graphicsLayer.add(polygonGraphic, 0);
```

      a) The "0" at the end is an index note. This means that the polygon will be brought to the front of the map, so that it can't be covered up by the FeatureLayer.

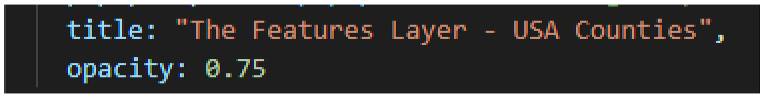
J. Open the HTML file in your browser, and make sure that there aren't any errors.
   1. You should see a large green rectangle hovering over the state of Wyoming.
   2. You should also be able to toggle the GraphicLayer on-and-off in the layer list widget that we added earlier.

VI. Adding Styling

A. The FeatureLayer and GraphicsLayer have additional styling options to personalize the map.
   1. These include the use of different colors, line styles, fonts, and more.

B. Let's begin styling with the FeatureLayer.
   1. Return to the code where the FeatureLayer object (countiesLayer) is.

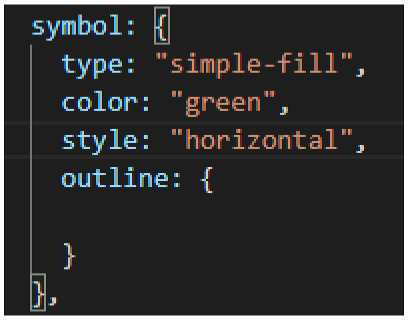
2. Beneath the popupTemplate property that you added while creating the widgets, add in the code from the image below.



```
title: "The Features Layer - USA Counties",
opacity: 0.75
```

   a) The "title" property lets you add a title for the layer that will appear in the layer list widget.
   b) The "opacity" property has a range from 0 to 1, with 0 being completely transparent, and 1 being solid.
      (1) Here, 0.75 will let the FeatureLayer be slightly translucent, enough so that you can see the cities and roads beneath the orange counties.
3. Open the HTML file in your browser.
   a) You should now see that the orange counties layer is translucent instead of solid.
   b) You should see the title that you have added in the layer list widget.

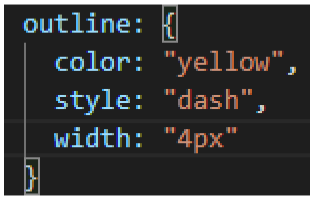
C. Next, let's style our shape over Wyoming in the GraphicsLayer.
   1. Return to the code where the GraphicsLayer is.
   2. In the polygon object, there is not anything to change with the geometry property. However, we can make some changes to the symbol property so that the shape has a different color and outline.
      a) Start by adding in the changes shown in the image below.



```
symbol: {
  type: "simple-fill",
  color: "green",
  style: "horizontal",
  outline: {

  }
},
```

         (1) The "style" property will change the way that the shape's fill is done.
            (a) In this case, the shape will be filled with horizontal green lines and a slightly translucent green background between the lines.
         (2) The "outline" property will be left blank for the moment, but it changes the styling for the borders of the shape.

b) The outline property has many properties, but for now, we will focus on the color, style, and width properties. Make your outline property match the image below.



```
outline: {
  color: "yellow",
  style: "dash",
  width: "4px"
}
```

(1) The "color" property defines what the border color of the shape will be.
  (a) In this case, our shape of Wyoming will have a yellow border around a green background.
(2) The "style" property defines the border style.
  (a) In this case, the border will be dashed. Instead of a solid line, it will be made of a series of yellow dashes.
(3) The "width" property defines how wide the border should be.
  (a) The default size is 1 pixel, so our shape will now have a border outline that is 4 times larger than it was before.
  (b) The width property also accepts pt units, too.

c) To summarize, we now have a polygon shape which hovers over the state of Wyoming, and that shape is filled in with green lines, and has a 4px, dashed, yellow border.

3. Open the HTML file in your browser.
   a) Provided that there are not any errors, you should see the styling changes that we made to the shape.
   b) If there are errors in the file, compare your file (arcgis-module.js) to the complete file (arcgis-module-complete.js) we have provided.