Writing a first widget for Experience Builder was a little overwhelming for me as a start. After getting a first understanding of React, I read and practiced the Tutorials section of https://developers.arcgis.com/experience-builder/guide/ several times and I decided to build my own tutorial based on the development of functional React components.

Most of the demands I received for an in-house widget are related to queries of services, so in this tutorial, I am looking at getting information on cities using the following service: https://sampleserver6.arcgisonline.com/arcgis/rest/services/SampleWorldCities/MapServer/0

The plan is to implement the widget in 3 stages:

- No interaction with any other widgets
- Interaction with a map widget
- Removing all hard-coded parameters (not completed yet….)

All widget development based on functional component starts with the following steps:

- Create a folder with the name of the widget you are making: Stage1
- Make a manifest.json file (do not forget to add: "dependency": ["jimu-arcgis"] as you are going to use some ArcGIS API for JavaScript modules
- Add an icon.svg file
- In the Stage1 folder, create a folder src with a folder runtime inside.
- In the runtime folder, make the widget.tsx file:

```
-   /** @jsx jsx */
-   import { AllWidgetProps, jsx } from "jimu-core";
-
-   export default function (props: AllWidgetProps<{}>) {
-
-     return (
-       <div>
-
-       </div>
-     );
-   }
```

Stage 1 will implement a Select component with a list of given cities and will display some parameters related to a selected city. For this, 3 inputs will be hard-coded: the url of the service, the list of the cities present in the Select component, the list of the attributes involved in the query.

```
  // Input section
  const queryUrl =
"https://sampleserver6.arcgisonline.com/arcgis/rest/services/SampleWorldCities/Ma
pServer/0";
  const cityList = ['Paris', 'San Francisco'];
  const attList = ["City_NAME", "POP", "POP_RANK"];
```

Then, create a Select component from the jimu-ui library that will display the list of the cities defined in the input section:

```
  return (
    <div>

      <Select placeholder="Select a City">
          {cityList.map((nam) => <option key={nam} value={nam}>{nam}</option>)}
      </Select>

    </div>
  );
```

Then, make the Select component operational by retrieving the selected value, saving it in the widget state (don’t forget to add the useState import) and handling the selection change:

```
  // Select section
  const [selectedCity, setSelectedCity] = useState("");
  const OnSelectedCity = e => {
    const myCity = e.target.value;
    console.log(myCity);
    setSelectedCity(myCity);
  };
```

Then, implement the query:

```
 // Query section
  let queryCity = new Query();
  queryCity.outFields = attList;
  queryCity.returnGeometry = false;
```

```
  const OnSelectedCity = e => {
    const myCity = e.target.value;
    console.log(myCity);
    setSelectedCity(myCity);
    queryCity.where =  attList[0] + " = '" + myCity + "'";
    query.executeQueryJSON(queryUrl, queryCity).then(function(results){
      const city = results.features[0];
      console.log(city.attributes[attList[1]]);

    });
  };
```

To display the results in a table, just add table entry in the return and keep the result of the query in the widget state. To make it nicer, make the rendering of the table conditioned to the existence of a selected city (for example by the && operator).

```
      {selectedCity != ""  &&   <section style={{padding : "10px"}}>
          <table>
            <tr>
              <th>City Name</th>
              <td>{cityInfo.nam}</td>
            </tr>
            <tr>
              <th>City Population</th>
              <td>{cityInfo.pop}</td>
            </tr>
            <tr>
              <th>City Rank</th>
              <td>{cityInfo.rnk}</td>
            </tr>
          </table>
      </section> }
```

For another cosmetic aspect, it is useful to clear the table when closing the widget. This can be done by listening to changes in the state of the widget.

```
  useEffect(() => {if(props.state == "CLOSED") {
    setSelectedCity("");
    }
   }, [props]);
```

Time to interact with the map! I followed the steps detailed in the tutorial “Get map coordinates” and adapted them to a functional component:

- In the widget root folder, create a config.json file that contains an empty object
- Create a folder setting in the src folder of the widget
- In the new folder, make the setting.tsx file that implements the MapWidgetSelector
- Modify the widget.tsx file to use the map object and the switch map tool
- Modify the query to zoom to the selected city
- Complete the clean-up when the widget closes. For this, I struggled a lot and finally came up with the following:
    - Keep in state the scale and center of the view when the widget opens, and the view is ready
    - Do the zoom to original view parameter when the widget closes

Stage 3 should remove all the hard-coded variables and deals with the case multiple-source map (right now if you switch view while the widget is open, the clean-up in the close is not ideal!).