Using React with the ArcGIS API for JavaScript

By Rene Rubalcava

The ArcGIS API for JavaScript is a powerful library that you can use to build applications that leverage the power of the ArcGIS platform. While you can use the ArcGIS API for JavaScript on its own to build compelling web mapping applications, some developers choose to integrate it with other JavaScript libraries and frameworks, especially when building larger web applications.

React is a popular open-source JavaScript library that is used for building reusable UI components for web applications. It is fast and simple and pairs really well with the ArcGIS API for JavaScript, which is why this implementation pattern is becoming popular among many developers.

This article gives you a brief overview of getting started using React with the ArcGIS API for JavaScript by examining a sample application. You may want to also check out the Using Frameworks topic in the online documentation for ArcGIS API for JavaScript to learn more about working with React and other libraries and frameworks.

Getting Started

View the source code for the application on GitHub (github.com/odoe/jsapi-react). For this application, we are going to use the @arcgis/webpack-plugin to help integrate the ArcGIS API for JavaScript into our application. A best practice is to isolate the work of the API from the UI components that you are going to build. This is a nice way to maintain a separation of concerns in your development.

In this example, we are going to do the work of creating our map in data/app.js, shown in Figure 1. Looking at this code snippet, you can see that this is where we create our map and view, but we do not attach our view to the page right away. We export a function called initialize that takes an argument for the container, which is a DOM element where our MapView will be displayed. This DOM element will come from the React component that we will write.

The Component

For the WebMap component, we are going to take advantage of a brand new feature in React called hooks that lets you use state and other React features without writing a class. Hooks are still a React proposal that is scheduled to be finalized in early 2019. I wouldn't recommend using them in production just yet, but I thought it would be fun to use them for this example. There are numerous React hooks you can use, but for our purposes, we are only concerned with two: useEffect and useRef.

```
import WebMap from "esri/WebMap";
import MapView from "esri/views/MapView";
import Search from "esri/widgets/Search";

const noop = () => {};

export const webmap = new WebMap({
  portalItem: {
    id: "974c6641665a42bf8a57da08e607bb6f"
  }
});

export const view = new MapView({
  map: webmap
});

export const search = new Search({ view });
view.ui.add(search, "top-right");

export const initialize = (container) => {
  view.container = container;
  view
    .when()
    .then(_ => {
      console.log("Map and View are ready");
    })
    .catch(noop);
  return () => {
    view.container = null;
  };
};
```

↑ Figure 1: data/app.js

The useEffect hook is run after the React component is rendered. This makes it perfectly suited for dynamically loading our data/app.js module and running the initialize function we created earlier. But how do we get the element for our component? That's where useRef comes in.

The useRef hook creates an object that exists for as long as the component is mounted. In our case, we want to keep track of the DOM element that is going to be created by our React component. We can see what this looks like in components/WebMap.js, shown in Figure 2.

```
import React, { useEffect, useRef } from "react";

export function WebMap() {
  const elementRef = useRef();

  useEffect(_ => {
    // lazy load the module that loads the JSAPI
    // and initialize it
    import("../data/app").then(
      app => app.initialize(elementRef.current)
    );
  });

  // assign elementRef to the ref of our component
  return (
    <div className="viewDiv" ref={elementRef}>
    </div>
  );
}
```

↖ Figure 2: components/WebMap.js

```
import "./config";

import React from "react";
import { render } from "react-dom";

import { WebMap } from "./components/WebMap";

const rootElement = document.getElementById("root");
render(<WebMap />, rootElement);
```

↙ Figure 3: index.js

Looking at this sample, you can see that we use the useEffect hook to lazy load the module that is responsible for handling the mapping portion of our application. This is a useful pattern that you can use in your applications to dynamically load the ArcGIS API for JavaScript in your webpack applications. Now we can render this component like any other React component in our application in index.js, as shown in Figure 3. In your finished application, a React component displays a WebMap, as shown in Figure 4.

Summary

We looked at how you can isolate the work of the ArcGIS API for JavaScript in its own module in your application and then dynamically load that module in your React components. We also looked at some of the cutting-edge uses of React hooks to help you build your React components with the API.

About the Author

Rene Rubalcava is an Esri software development engineer, blogger, author, geodev, and connoisseur of programming languages and JavaScript frameworks. Follow him on Twitter @odoenet, and read his blog at odoe.net/blog.

↓ Figure 4: In your finished application, a React component displays a WebMap.