**Pre-requisite**

* NODE.JS [LATEST VERSION]
* Visual Studio Code

**Step 1: Base scaffolding**

* Create a SFDX Project with manifest in Visual studio code.
* In the terminal run the below, to initialize the package.json
  + **npm init -y**
* Then, run the below command to create a public folder and create a file manually, called “index.html” [ public/index.html]
  + **mkdir public**
* Paste the below code in the index.html file,

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>React + Webpack</title>**

**</head>**

**<body>**

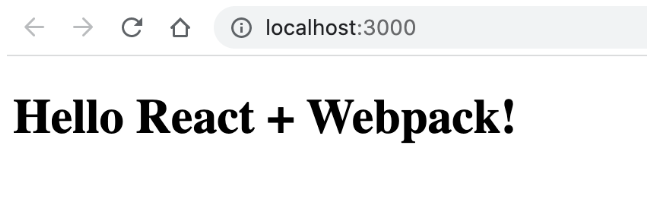
**<h1>Hello React + Webpack! </h1>**

**</body>**

**</html>**

* Now run the below command to view the result of the index.html in the localhost
* **npx serve public**

And it does! If you navigate to http://localhost:3000, you should see the hello message we just added:



**Step 2: Adding Webpack**

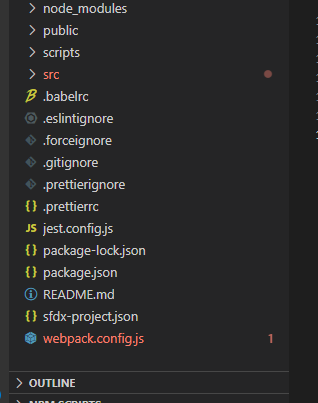
* First, install Webpack:
  + **npm install webpack webpack-cli --save-dev**
* Next, let's create a simple JavaScript file that we can configure Webpack to bundle:
  + **mkdir src**
* create a file manually in the src folder, called “index.js” [src/index.js] and paste below code in the file.

**const helloDiv = document.createElement("div");**

**helloDiv.innerHTML = "Hello from Javascript!";**

**document.body.append(helloDiv);**

* Then, we need to configure Webpack by creating a webpack.config.js file in the root of the project and paste the below code.



**// webpack.config.js**

**const path = require("path");**

**module.exports = {**

**entry: "./src/index.js",**

**output: {**

**filename: "main.js",**

**path: path.resolve(\_\_dirname, "build"),**

**},**

**};**

* Finally, in package.json, add a new “build” Script

**"scripts": {**

**"build": "webpack"**

**},**

Now, let's try it out! After running npm run build, you should see a new folder was created, called “build”, with a “main.js” file in it!

Next, we need to move the static assets to the bundle.

More specifically, we want to also include the index.html file in the build folder.

Easiest way to do this is with the HtmlWebpackPlugin.

* + **npm install html-webpack-plugin --save-dev**

And update the webpack.config.file:

const path = require("path");

**const HtmlWebpackPlugin = require("html-webpack-plugin");**

**module.exports = {**

**entry: "./src/index.js",**

**output: {**

**filename: "main.js",**

**path: path.resolve(\_\_dirname, "build"),**

**},**

**plugins: [**

**new HtmlWebpackPlugin({**

**template: path.join(\_\_dirname, "public", "index.html"),**

**}),**

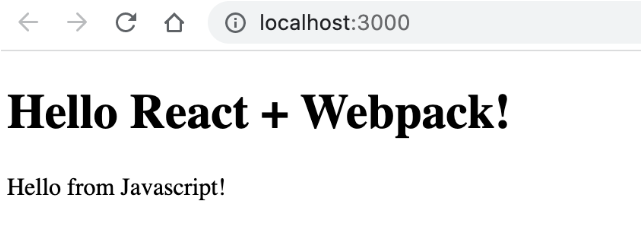
**],**

**};**

Now run the following command in the terminal

* + **npm run build**
  + **npx serve build**

And it works! You should now also see the message "Hello from Javascript"



**Adding Webpack dev server**

So far, it was ok to just use npx serve to check our app works, but in real life, it's easier to just use the webpack-dev-server, so let's add that as well.

**npm install --save-dev webpack-dev-server**

Then, configure it in the Webpack config:

**{**

**// ...,**

**devServer: {**

**static: {**

**directory: path.join(\_\_dirname, "build"),**

**},**

**port: 3000,**

**}**

**}**

and then add npm run start script to package json; and while we're there, pass in the right "mode":

**{**

**// ...,**

**"scripts": {**

**"build": "webpack --mode production",**

**"start": "webpack serve --mode development"**

**}**

**}**

Finally, check that it works: run npm run start, open http://localhost:3000 and check the app still works as before.

**Step 3: Adding Babel**

This is useful for allowing us to use all ES6 features and having them transpiled down to JS versions that all browsers can understand.

First, let's install the required packages:

**npm i @babel/core @babel/preset-env babel-loader --save-dev**

Next, update the Webpack config to tell it to pass the files through Babel when bundling:

**{**

**// ...,**

**module: {**

**// exclude node\_modules**

**rules: [**

**{**

**test: /\.(js)$/,**

**exclude: /node\_modules/,**

**use: ["babel-loader"],**

**},**

**],**

**},**

**// pass all js files through Babel**

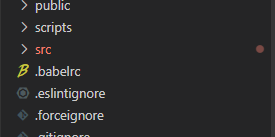
**resolve: {**

**extensions: ["\*", ".js"],**

**}**

**}**

Then, create the Babel config file - .babelrc. This is where we configure Babel to apply the preset-env transform.



**// .babelrc**

**{**

**"presets": [**

**"@babel/preset-env"**

**]**

**}**

Optionally, update the index.js to contain some ES6 features that wouldn't work without Babel

**// Use a feature that needs Babel to work in all browsers :)**

**// arrow functions + Array fill**

**const sayHelloManyTimes = (times) =>**

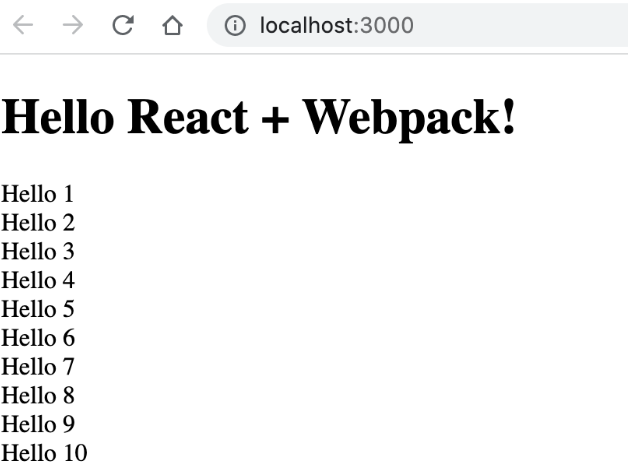
**new Array(times).fill(1).map((\_, i) => `Hello ${i + 1}`);**

**const helloDiv = document.createElement("div");**

**helloDiv.innerHTML = sayHelloManyTimes(10).join("<br/>");**

**document.body.append(helloDiv);**

Finally, let's check everything works - run npm run start and check the app correctly runs:



**Step 4: Add React**

Finally, we can add React

First, install it:

* **npm i react react-dom --save**
* **npm i @babel/preset-react --save-dev**

Then, update the .babelrc file to also apply the preset-react transform. This is needed, among other things, to support JSX.

**// .babelrc**

**{**

**"presets": [**

**"@babel/preset-env",**

**["@babel/preset-react", {**

**"runtime": "automatic"**

**}]**

**]**

**}**

Also, we need to update the Webpack config to pass jsx files through Babel as well:

**// webpack.config.js**

**{**

**// ...,**

**module: {**

**// exclude node\_modules**

**rules: [**

**{**

**test: /\.(js|jsx)$/, // <-- added `|jsx` here**

**exclude: /node\_modules/,**

**use: ["babel-loader"],**

**},**

**],**

**},**

**// pass all js files through Babel**

**resolve: {**

**extensions: ["\*", ".js", ".jsx"], // <-- added `.jsx` here**

**},**

**}**

let's create a React component, Create a file “Hello.js” in the src folder and paste the below code.

**// src/Hello.js**

**const Hello = () => <h1>Hello from React!</h1>;**

**export default Hello;**

And add it to the main app file:

**// index.js**

**import React from "react";**

**import { createRoot } from "react-dom/client";**

**import Hello from "./Hello";**

**const container = document.getElementById("root");**

**const root = createRoot(container);**

**root.render(<Hello />);**

Finally, we also update the index.html to provide a "root" node for the app:

**<!-- index.html -->**

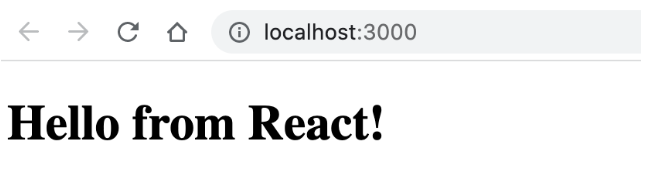
**<!-- ... -->**

**<body>**

**<div id="root"></div>**

**</body>**

Let's check that it works - run npm run start and you should see "Hello from React!":



Also, check there are no errors in the console.

You can also check that the app correctly runs in production, by running npm run build and then npx serve build.

**Step 5: Build the react component into a Static resource:**

* In the output module of the webpack.config.js, change the path as below

**output: {**

**filename: "main.js",**

**path: path.resolve(\_\_dirname, "force-app/main/default/staticresources/reactComponent")**

**},**

Run the command npm run build.

This will build a static resource with the name reactComponent.

Now create a meta.xml file for the static resource as below and deploy it to the SF org.

**reactComponent.resource-meta.xml**

**<?xml version="1.0" encoding="UTF-8"?>**

**<StaticResource xmlns="**[**http://soap.sforce.com/2006/04/metadata"**](http://soap.sforce.com/2006/04/metadata%22)**>**

**<cacheControl>Public</cacheControl>**

**<contentType>application/zip</contentType>**

**</StaticResource>**

Step 6: Loading REACTJS component in Salesforce.

**ReactJS with Aura component**

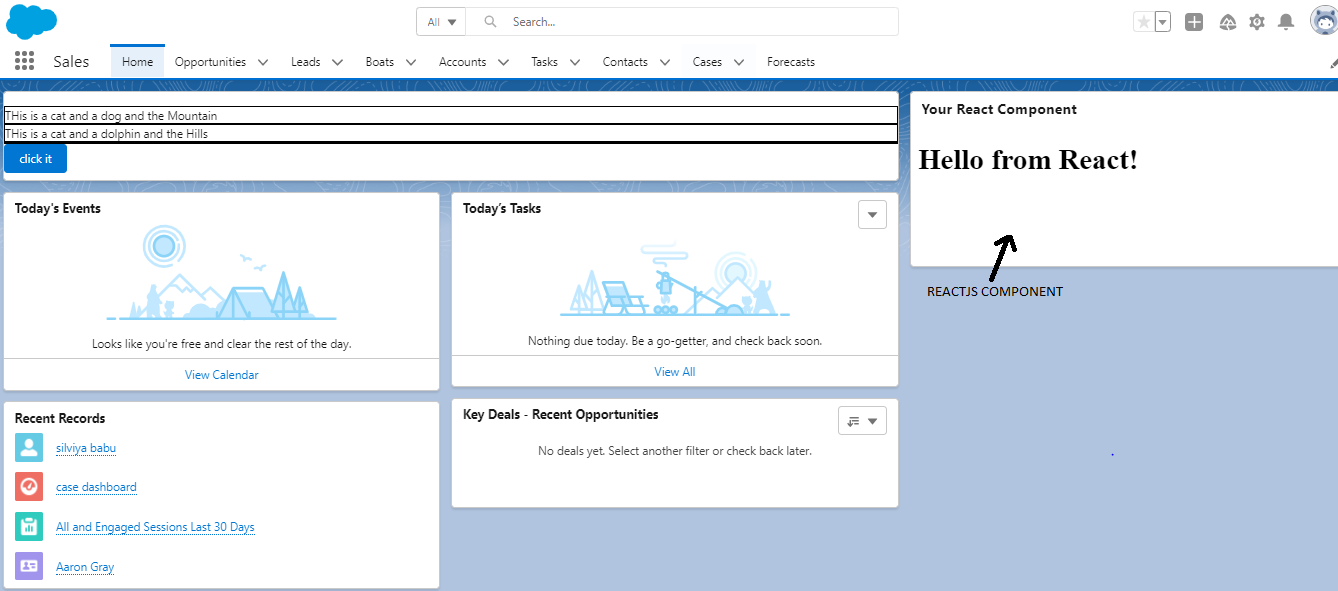
In the Aura Component, using the <lightning: container>, it is possible to load a third-party resource into the page.

<lightning:card title="Your React Component">

<lightning:container aura:id="jsApp" src="{!$Resource.reactComponent + '/index.html'}"/>

</lightning:card>

The above code will render the REACTJS Code into Aura Component.



---------------THE END----------------