Bundling assets

Earlier, you learned what assets are in React and the best practices for storing them in your project folders.

In this reading, you will learn about the advantages and disadvantages of embedding assets, including examples of client/server-side assets. You will also learn about the trade-offs inherent in using asset-heavy apps.

The app's files will likely be bundled when working with a React app. Bundling is a process that takes all the imported files in an app and joins them into a single file, referred to as a **bundle**. Several tools can perform this bundling. Since, in this course, you have used the `create-react-app` to build various React apps, you will focus on webpack. This is because webpack is the built-in tool for the `create-react-app`.

Let's start by explaining what webpack is and why you need it.

Simply put, webpack is a module bundler.

Practically, this means that it will take various kinds of files, such as SVG and image files, CSS and SCSS files, JavaScript files, and TypeScript files, and it will bundle them together so that a browser can understand that bundle and work with it.

Why is this important?

When building websites, you could probably do without webpack since your project's structure might be straightforward: you may have a single CSS library, such as Bootstrap, loaded from a CDN (content delivery network). You might also have a single JavaScript file in your static HTML document. If that is all there is to it, you do not need to use webpack in such a scenario.

However, modern web development can get complex.

coach

Here is an example of the first few lines of code in a single file of a React application:

```
import React from 'react';
import '@atlaskit/css-reset';
import styled from 'styled-components';
import './index.css';
import { ThemeProvider } from './contexts/theme';
import { DragDropContext } from 'react-beautiful-dnd';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Nav from './components/Nav';
import data from './data';
import Loading from './components/Loading';

```

The imports here are from fictional libraries and resources because the specific libraries are not necessary. All these different imports can be of various file types: .js, .svg, .css, and so on.

In turn, all the imported files might have their own imported files, and even those might have their imports.

This means that depending on other files, all of these files can create a **dependency graph**. The order in which all these files are loading is essential. That dependency graph can get so complex that it becomes almost impossible for a human to structure a complex project and bundle all those dependencies properly.

This is the reason you need tools like webpack.

So, webpack builds a dependency graph and bundles modules into one or more files that a browser can consume.

While it is doing that, it also does the following:

- It converts modern JS code - which can only be understood by modern browsers - into older versions of JavaScript so that older browsers can understand your code. This process is known as *transpiling*. For example, you can transpile ES7 code to ES5 code using webpack.
- It optimizes your code to load as quickly as possible when a user visits your web pages.
- It can process your SCSS code into the regular CSS, which browsers can understand.
- It can build source maps of the bundle's building blocks
- It can produce various kinds of files based on rules and templates. This includes HTML files, among others.

coach

Another significant characteristic of webpack is that it helps developers create modern web apps.

It helps you achieve this using two modes: **production** mode or **development** mode.

In **development** mode, webpack bundles your files and optimizes your bundles for updates - so that any updates to any of the files in your locally developed app are quickly re-bundled. It also builds source maps so you can inspect the original file included in the bundled code.

In **production** mode, webpack bundles your files so that they are optimized for speed. This means the files are minified and organized to take up the least amount of memory. So, they are optimized for speed because these bundles are fast to download when a user visits the website online.

Once all the source files of your app have been bundled into a single bundle file, then that single bundle file gets served to a visitor browsing the live version of your app online, and the entire app's contents get served at once.

This works great for smaller apps, but if you have a more extensive app, this approach is likely to affect your site's speed. The longer it takes for a web app to load, the more likely the visitor will leave and move on to another unrelated website. There are several ways to tackle this issue of a large bundle.

One such approach is code-splitting, a practice where a module bundler like webpack splits the single bundle file into multiple bundles, which are then loaded on an as-needed basis. With the help of code-splitting, you can **lazy load** only the parts that the visitor to the app needs to have at any given time. This approach significantly reduces the download times and allows React-powered apps to get much better speeds.

There are other ways to tackle these problems.

An example of a viable alternative is SSR (Server-side rendering).

With SSR, React components are rendered to HTML on the server, and the visitor downloads the finished HTML code. An alternative to SSR is client-side rendering, which downloads the index.html file and then lets React inject its own code into a dedicated HTML element (the **root** element in `create-react-app`). In this course, you've only worked with client-side rendering.

Sometimes, you can combine client-side rendering and server-side rendering. This approach results in what's referred to as **isomorphic apps**.

In this reading, you learned about the advantages and disadvantages of embedding assets, including examples of client/server-side assets. You also learned about the trade-offs inherent in the use of asset-heavy apps.

Mark as completed

Like Dislike Report an issue

coach