Modular Applications

1. The need for modularity
2. Example using Webpack

1. The Need for Modularity

- The problem
- The solution - Webpack
- Installing Webpack
- What does Webpack do?

The Problem

- Your web applications are likely to become quite large
  - 100's or 1000's of components
  - Various 3rd-party JS and CSS libraries
  - Lots of versions of the above
- React introduces some additional considerations
  - How do you transpile ES6 and JSX into ES5?
- How do you manage and coordinate all of this?

The Solution - Webpack

- Various toolsets have cropped up over the years, to help you manage all your files and processes
  - E.g. Gulp, Grunt, Browserify
- In recent years, Webpack has emerged as the preferred tool for bundling CommonJS modules
  - Node.js introduced the concept of CommonJS modules
  - ES6 supports CommonJS modules too
  - React apps consist of a bunch of CommonJS modules

Installing Webpack

- You can install `webpack` using npm, as follows:

```
npm install webpack -g
```

What does Webpack do?

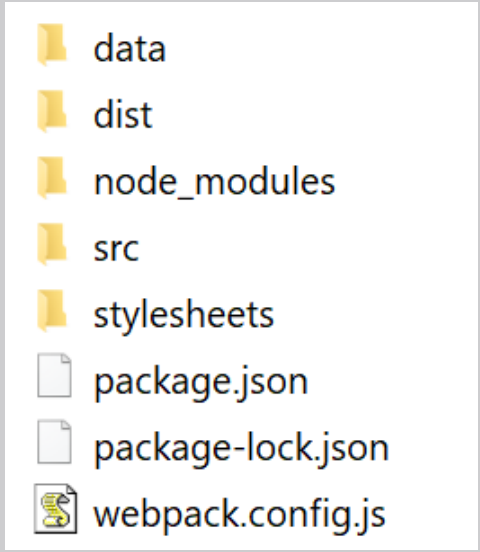
- Webpack is a module bundler
  - It bundles all your source files into a single file
- Benefits:
  - Dramatically improves network performance
  - Browser can download your app in a single HTTP request
- Additional cool capabilities in Webpack:
  - Code minification, uglification, hot module replacement

2. Example using Webpack

- Overview
- Defining a `package.json` file
- Installing packages
- Configuring Webpack
- App structure
- Packaging and running the app
- Another example

Overview

- In this section we'll take a look at a simple React app that uses Webpack, see the following folder:
  - `Demo1-SimpleComponents`



Defining `package.json` (1 of 4)

- `package.json` looks like this
  - See following slides for details

```
{
  "name": "retailersDirectory-app",
  "version": "1.0.0",
  "description": "Modular app via webpack",
  "main": "index.js",

  "dependencies": { … },
  "devDependencies": { … },
  "scripts": { … }
}
```

Defining `package.json` (2 of 4)

- We define the following dependencies:

```
"dependencies": {
   "react": "16.13.1",
   "react-dom": "16.13.1"
}
```

Defining `package.json` (3 of 4)

- We define the following dev tools, amongst others

```
"devDependencies": {
   "@babel/core": "7.9.6",
   "babel-loader": "8.1.0",
   "@babel/preset-env": "7.9.6",
   "@babel/preset-react": "7.9.4",
   "webpack": "4.43.0",
   "webpack-cli": "3.3.11"
   …
}
```

Defining `package.json` (4 of 4)

- We define the following script
  - Runs `webpack` to package our application
  - Then concurrently runs `webpack` and `live-server`
  - If we edit a source file, it will be rebundled and reloaded

```
"scripts": {
   "start": "webpack --progress &&
             concurrently
             \"webpack --progress --watch\"
             \"live-server dist\""
}
```

Installing Packages

- To install packages, open a command window in your project folder, and run the following command:

```
npm install
```

- npm downloads specified packages and dependencies
  - See the `node_modules` sub-folder

Configuring Webpack

- To configure how Webpack performs its packaging and bundling, see `webpack.config.js`
  - We package app in dev mode (prevents minification)
  - Application entry point is `src/index.js`
  - Bundled output will be in `dist/assets/bundle.js`

- Babel does a lot of cool things, via "presets"
  - `@babel/preset-env` - Compiles code to ES5
  - `@babel/preset-react` - Compiles JSX

Application Structure

- `src\index.js`
  - Entry point for our code
- `src\components`
  - Defines our React components, e.g. one per file
- `data`
  - Contains the data for our app, loaded into app on startup
- `dist\index.html`
  - Entry point for our web app

Packaging and Running the Application

- To package and run the app:

```
npm start
```

- What happens:
  - Builds and bundles the app (into the `dist` folder)
  - Starts `live-server`
  - Opens a browser and loads your home page
  - If you change any file, it's rebundled and reloaded

Another Example

- We have another example, in the following folder
  - `Demo2-ComponentHierarchy`
- What's different in this example
  - Uses more "interesting" data
  - Contains more components

Summary

- The need for modularity
- Example using Webpack