Web Front-End Development
Week 8: JavaScript Modules and NPM

JavaScript programs started off as small scripts that provided a little interactivity in web pages. But now we have entire applications running in browsers powered by JavaScript and as JavaScript programs have grown many lack structure which makes them hard to keep organized and maintain.

JavaScript Modules

JavaScript modules are a way to break programs into pieces that are easier to maintain, easier to read, and are reusable. A module specifies the other modules it relies on through its dependencies. And modules can make its functionality available for other modules to use through its interface.

Up until 2015 JavaScript had no built-in module system. CommonJS was created in 2009 and started working with JavaScript in 2011.

In 2015 the JavaScript standard introduce its own, and of course different, module system, ES (ECMAScript) modules as part of ES6. ESModules were adopted by all major browsers by 2018.

Everything declared inside a module is local to the module by default. To make something declared in a module public so other modules can use it you must export that feature using the `export` keyword. You can export any top-level function, class, var, let, or const.

To use those functions or classes in other files you need to import them using the `import` keyword. Once imported you can use them in your code.

ES module imports happen before a module's script starts running. That means `import` declarations must always be at the top of the file before any other code and can't be inside functions or blocks. The names of dependencies must be quoted strings and must include the relative path (./ for example).

- . stands for the current directory, so ./filename will look for the file in your current directory
- ../ stands for one directory up
- / stands for the root of the file system

You must add `type="module"` to your script tags for any files that use import or export. This ensures that the code gets loaded as a module.
Module scripts are *always* deferred so it has the same effect as the `defer` attribute.

Modules must be used on a web server, either local or on the internet, not loaded as a file (file:///).

Example:
https://replit.com/@aileenjp/Canvas-module
Let's look at a simple example of using a module.

I have canvas.js with a smile() function that draws a smile on the canvas of the ID passed into the function. I want this function to be available in my other scripts so I export it.

```
export function smile(canvasID) { … }
```

Then in script.js I import smile so I can use the function in this script.

```
import {smile} from './canvas.js'
```

Note that I use ./ to indicate current directory.
Then I update my html to script.js and since it's a module I need to have that as the type.

```
<script type="module" src="script.js">
```

Packages
A package bundles together modules as a way to ensure that you have the dependency modules as well. They also include documentation so developers know how to use the modules in a package.

NPM
https://www.npmjs.com/
The infrastructure to store and distribute packages is provided by the Node Package Manager (NPM). Node.js and most packages on NPM use CommonJS modules. Node 14 now supports ESModules as well.
NPM has two parts:

- An online registry where you can upload and download packages
    - Many packages are published under a license that allows other people to use it but some require you to also publish code that you build on top of the package under the same license.
- A program, bundled with Node.js, that helps you install them

NPM comes pre-installed with Node.js and you run it in the command line with `npm`.
`npm init` creates a `package.json` file that has information about the package you install.
`npm install <package name>` installs a package published on the NPM registry.
After running `npm install` NPM will have created a directory called `node_modules` and in `node_modules` you will see a directory for the package you just installed.

Another popular package manager is Yarn (Yet Another Resource Negotiator), created by Facebook.
You'll need to install Yarn:

```
npm install yarn
```

I'm not going to cover the differences between the two, you can use either one.

Node.js
https://nodejs.org/en/
Node.js is an open-source and cross-platform runtime environment for executing JavaScript code outside a browser. Along with NPM it comes with some built-in modules such as fs (file system), and http (can run a http server and make http requests).

Bundlers
Programs that use modules and their dependencies can take time to request and load possibly hundreds of files. Bundlers roll all the required modules into a single file that you can use in your project when you publish it to the Web.
Popular bundlers include Webpack, Browserify, and rollup.
They usually start with an entry file and then look for all the dependencies of your code, such as the imported modules, and packs them together in a single file that your application can use (bundle.js).
Webpack https://webpack.js.org/
To install Webpack:

```
npm install webpack
```

Firebase Example:
https://creative.colorado.edu/~apierce/samples/firebase_orders_cfs9.html
We'll use the Firebase SDK version 9 as an example of using modules in our project.
https://firebase.google.com/docs/web/modular-upgrade?authuser=0
Compat
For migration purposes Firebase has made the version 9 libraries available through a CDN so we can refactor our code using the new functions before moving to a fully modular style.
Remove all Firebase code from your HTML.
In your JavaScript file the links to the CDN will now be import statements.

```
  import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.1.2/firebase-app.js";
  import { getFirestore, collection, onSnapshot } from
"https://www.gstatic.com/firebasejs/9.1.2/firebase-
firestore.js";
```

Copy your firebaseConfig into your JavaScript file.
Update to use the new V9 functions.

```
V8: firebase.initializeApp(firebaseConfig);
V9: const app = initializeApp(firebaseConfig);

V8: let database = firebase.firestore();
V9: const db = getFirestore(app);


V8: let order_list = document.getElementById("orders");
V9: const orders = collection(db, "orders");


V8: database.collection("orders").onSnapshot((querySnapshot) =>
{…}
V9: onSnapshot(orders, (querySnapshot) => {…}
```

In the HTML the link to your JavaScript file now must contain type="module".

```
<script type="module" src="js/firebase_orders_cfs9.js"></script>
```

Modular
Now we'll set up our project to install the Firebase packages from NPM.
https://firebase.google.com/docs/web/module-bundling

You'll need to install Node.js if you don't already have it installed. https://nodejs.org/en/
Terminal
Check node version:
```
node -v
```

You can upgrade node using n or nvm (will need to install those first, using npm)
```
npm install -g n
sudo n stable
```
(enter password)

Check version of npm
```
npm -v
```

Update npm to the latest version
```
npm install npm -g
check permissions missing write access error
```
fix:
```
npm set prefix ~/.npm
PATH="$HOME/.npm/bin:$PATH"
PATH="./node_modules/.bin:$PATH"
source ~/.bashrc
```

Create a directory where you're going to install Firebase.
```
cd Firebase
```
Initialize npm which will walk you through creating a package.json file.
```
npm init
```

```
package name: (firebase) firebase-buff-tickets
version: (1.0.0)
description: firebase buff tickets
entry point: (index.js) js/firebase_orders_cfs9module.js
test command:
git repository:
keywords:
author: Aileen Pierce
license: (ISC)
```

Your package name would be used as an identifier if you published it. So it should be different than the package/modules you will be installing.

Entry point/main: the JavaScript file you want to include. This should be the “top level” js file, so it might include other imports.

Add the Firebase package (modules) as a dependency
`npm i firebase`

`(look at package.json file now)`

Install webpack (or other bundler)
`npm i webpack webpack-cli -D`

Create a file at the root of your local project named `webpack.config.js`
Make sure your entry point matches what you entered for main in package.json with ./ at the beginning.

```
const path = require('path');

module.exports = {
  // The entry point file described above
  entry: './js/firebase_orders_cfs9module.js',
  // The location of the build folder described above
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  // Optional and for development only. This provides the ability to
  // map the built code back to the original source format when debugging.
  devtool: 'eval-source-map',
};
```

Add a npm script to run the webpack build. Add the following key/value pair to the “scripts” object in package.json.

```
"scripts": {
    "build": "webpack --mode=development"
  },
```

Create a runtime build environment
`npm run build`

Now you will have a “dist” directory for distribution. In this directory is bundle.js which is the bundled file with all the needed JavaScript.
Update your HTML to link to this JavaScript file.
Upload your HTML, CSS, and dist folder to a server (or run localhost) to test.
Every change to your JavaScript file will require a new build (npm run build) to create a new bundle.js file.

https://creative.colorado.edu/~apierce/samples/firebase-buff-tickets/firebase_orders_cfs9module.html
(link to source at the bottom of the page)