Code Organization with ES6 Modules

In this lesson, we'll discuss the best practices of React specific code organization on a local setup for a large React application i.e., the most efficient way to put components into modules (files) and modules into folders.

File structure: Best Practices

You might wonder why we didn't follow the best practices of putting components into different files for the *App.js* file. We already have multiple components in the file that can be defined in their own files/folders (modules). For the sake of learning React, it is practical to keep these items in one place. Once your React application grows, you should consider splitting these components into multiple modules so it can scale properly.

In the following, I propose several module structures you can apply. You can apply them as an exercise later, but we will continue developing our app with the *App.js* file to keep things simple. Most React setups have a *src* folder though which contains the *App.js* file.

Module Structure #1: Naming by component

Here is one possible module structure:

```
src/
  index.js
  index.css
  App.js
  App.test.js
  App.css
  Button.js
  Button.test.js
  Button.css
  Table.js
  Table.test.js
  Table.css
  Search.js
  Search.test.js
  Search.css
```

This one separates the components into their own files, but it doesn't look too

promising. You can see a lot of naming duplications, and only the file extension differs.

Module Structure #2: Categorizing by Component

Another module structure could be:

```
src/
  index.js
  index.css
  App/
    index.js
    test.js
    index.css
  Button/
    index.js
    test.js
    index.css
  Table/
    index.js
    test.js
    index.css
  Search/
    index.js
    test.js
    index.css
```

Now it looks cleaner than before. The index naming of a file describes it as an entry point file to the folder. It is just a common naming convention, but you can use your own naming as well. In this module structure, a component is defined by its component declaration in the JavaScript file, but also by its style and tests. If you have trouble finding your components this way while searching for them in your editor/IDE, search for paths rather than files (e.g. search for “Table index”).

Another step is extracting the constant variables from the App component, which were used to compose the Hacker News API URL.

```
src/
  index.js
  index.css
  constants/
    index.js
  components/
    App/
      index.js
      test.js
      index.css
    Button/
      index.js
      test.js
```

```
      index.css
    ...
```

Naturally, the modules would split up into *src/constants/* and *src/components/*. The *src/constants/index.js* file could look like the following:

```
export const DEFAULT_QUERY = 'redux';
export const DEFAULT_HPP = '100';
export const PATH_BASE = 'https://hn.algolia.com/api/v1';
export const PATH_SEARCH = '/search';
export const PARAM_SEARCH = 'query=';
export const PARAM_PAGE = 'page=';
export const PARAM_HPP = 'hitsPerPage=';
```

The *App/index.js* file could import these variables in order to use them.

```
import {
  DEFAULT_QUERY,
  DEFAULT_HPP,
  PATH_BASE,
  PATH_SEARCH,
  PARAM_SEARCH,
  PARAM_PAGE,
  PARAM_HPP,
} from '../../constants/index.js';

...
```

When you use the *index.js* naming convention, you can omit the filename from the relative path.

```
import {
  DEFAULT_QUERY,
  DEFAULT_HPP,
  PATH_BASE,
  PATH_SEARCH,
  PARAM_SEARCH,
  PARAM_PAGE,
  PARAM_HPP,
} from '../../constants';

...
```

The *index.js* file naming is convention was introduced in the Node.js world, where the index file is the entry point to a module. It describes the public API to the module. External modules are only allowed to use the *index.js* file to

import shared code from the module. Consider the following module structure to demonstrate it:

```
src/
  index.js
  App/
    index.js
  Buttons/
    index.js
    SubmitButton.js
    SaveButton.js
    CancelButton.js
```

The *Buttons/* folder has multiple button components defined in its distinct files. Each file can `export default` the specific component, making it available to *Buttons/index.js*. The *Buttons/index.js* file imports all different button representations and exports them as public module API.

```
import SubmitButton from './SubmitButton';
import SaveButton from './SaveButton';
import CancelButton from './CancelButton';

export {
  SubmitButton,
  SaveButton,
  CancelButton,
};
```

Now the *src/App/index.js* can import the buttons from the public module API located in the *index.js* file.

```
import {
  SubmitButton,
  SaveButton,
  CancelButton
} from '../Buttons';
```

However, it can be seen as bad practice to reach into other files than the *index.js* in the module, as it breaks the rules of encapsulation.

```
// bad practice, don't do it
import SubmitButton from '../Buttons/SubmitButton';
```

We refactored the source code in a module with the constraints of encapsulation. Again, we'll keep things simple in this course, but you should always refactor your source code to keep it clean.

Exercises

- Refactor your *src/App.js* file into multiple component modules when you finished the course on a local React setup