# React Project Structure

Let's learn about the structure of a react project.

**We'll cover the following**



With multiple React components in one file, you might wonder why we didn't put components into different files for the *src/App.js* file from the start. We already have multiple components in the file that can be defined in their own files/folders (also called modules). For learning, it's more practical to keep these components in one place. Once our application grows, we'll consider splitting these components into multiple modules so it scales properly.

Before we restructure our React project, recap JavaScript's import and export statements. Importing and exporting files are two fundamental concepts in JavaScript you must learn before React. There's no right way to structure a React application, as they evolve naturally along with the project's structure.

We'll complete a simple refactoring for the project's folder/file structure for the sake of learning about the process. Afterward, there will be a few additional options about restructuring this project or React projects in general. You can continue with the restructured project, though we'll continue developing with the *src/App.js* file to keep things simple.

On the command line in your project's folder, navigate into the *src/* folder and create the following component dedicated files:

```
cd src
touch List.js InputWithLabel.js SearchForm.js
```

> **Note:** This command is to be executed when you run this on your local machine. You can see the live execution of code at the end of this lesson.

Move every component from the *src/App.js* file in its own file, except for the List component which has to share its place with the Item component in the *src/List.js* file. Then in every file make sure to import React and to export the component which needs to be used from the file. For instance, in *src/List.js* file:

```
import React from 'react';

const List = ({ list, onRemoveItem }) =>
  list.map(item => (
    <Item
      key={item.objectID}
      item={item}
      onRemoveItem={onRemoveItem}
    />
  ));

const Item = ({ item, onRemoveItem }) => (
  <div>
    <span>
      <a href={item.url}>{item.title}</a>
    </span>
    <span>{item.author}</span>
    <span>{item.num_comments}</span>
    <span>{item.points}</span>
    <span>
      <button type="button" onClick={() => onRemoveItem(item)}>
        Dismiss
      </button>
    </span>
  </div>
);


export default List;
```

src/List.js

Since only the List component uses the Item component, we can keep it in the same file. If this changes because the Item component is used elsewhere, we can give the Item component its own file. The SearchForm component in the *src/SearchForm.js* file must import the InputWithLabel component. Like the Item component, we could have left the InputWithLabel component next to the SearchForm; but our goal is to make InputWithLabel component reusable with other components. We'll probably import it eventually.

```
import React from 'react';

import InputWithLabel from './InputWithLabel';

const SearchForm = ({
  searchTerm,
  onSearchInput,
```

```
  onSearchSubmit,
}) => (
  <form onSubmit={onSearchSubmit}>

    <InputWithLabel
      id="search"
      value={searchTerm}
      isFocused
      onInputChange={onSearchInput}
    >
      <strong>Search:</strong>
    </InputWithLabel>

    <button type="submit" disabled={!searchTerm}>
      Submit
    </button>
  </form>
);

export default SearchForm;
```

src/SearchForm.js

The App component has to import all the components it needs to render. It doesn't need to import InputWithLabel, because it's only used for the SearchForm component.

```
import React from 'react';
import axios from 'axios';

import SearchForm from './SearchForm';
import List from './List';

...

const App = () => {
  ...
};

export default App;
```

src/App.js

Components that are used in other components now have their own file. Only if a component (e.g. Item) is dedicated to another component (e.g. List) do we keep it in the same file. If a component should be used as a reusable component (e.g. InputWithLabel), it also receives its own file. From here, there are several strategies to structure your folder/file hierarchy. One scenario is to create a folder for every component:

```
- List/
-- index.js
- SearchForm/
```

```
-- index.js

- InputWithLabel/
-- index.js
```

The *index.js* file holds the implementation details for the component, while other files in the same folder have different responsibilities like styling, testing, and types:

```
- List/
-- index.js
-- style.css
-- test.js
-- types.js
```

If using CSS-in-JS, where no CSS file is needed, one could still have a separate *style.js* file for all the styled components:

```
- List/
-- index.js
-- style.js
-- test.js
-- types.js
```

Sometimes we'll need to move from a **technical-oriented folder structure** to a **domain-oriented folder structure**, especially once the project grows. Universal *shared/* folder is shared across domain specific components:

```
- Messages.js
- Users.js
- shared/
-- Button.js
-- Input.js
```

If you scale this to the deeper level folder structure, each component will have its own folder in a domain-oriented project structure as well:

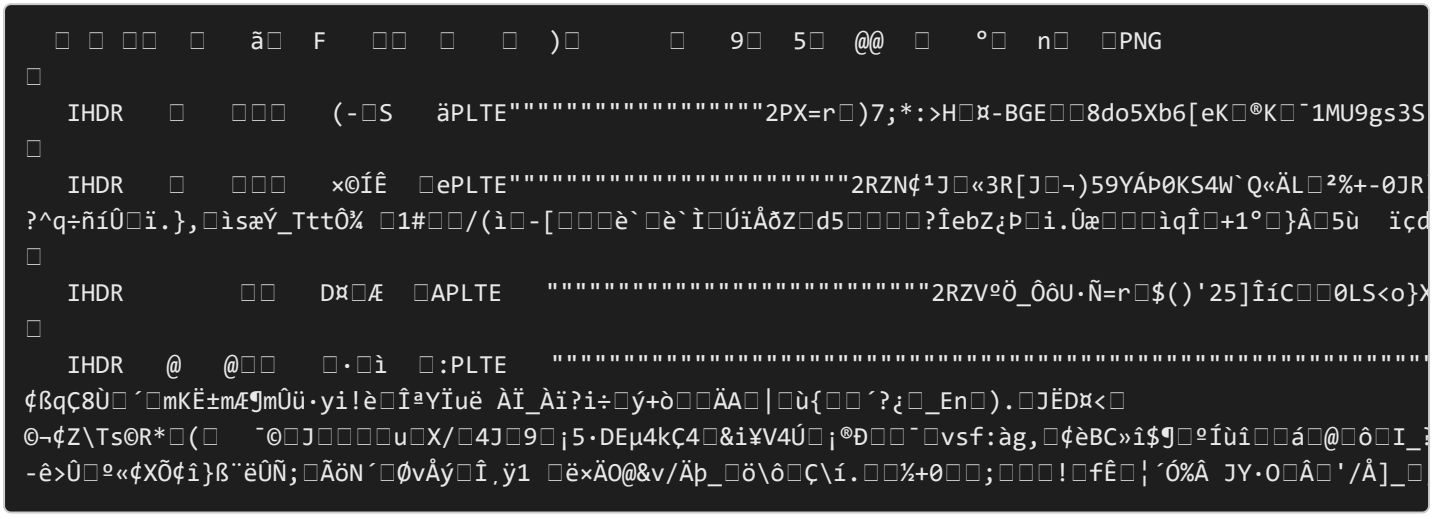
```
- Messages/
-- index.js
-- style.css
-- test.js
-- types.js
```

```
- Users/
-- index.js

-- style.css
-- test.js
-- types.js
- shared/
-- Button/
--- index.js
--- style.css
--- test.js
--- types.js
-- Input/
--- index.js
--- style.css
--- test.js
--- types.js
```

There are many ways on how to structure your React project from small to large project: simple to complex folder structure; one-level nested to two-level nested folder nesting; dedicated folders for styling, types and testing next to implementation logic. There is no right way for folder/file structures.

A project's requirements evolve over time and so should its structure. If keeping all assets in one file feels right, then there is no rule against it. Just try to keep the nesting level shallow, otherwise you could get lost deep in folders.



## Exercises: #

- Confirm the changes from the last section.
- Read more about JavaScript's import and export statements.
- Keep the current folder structure if you feel confident. The ongoing sections will omit it, only using the *src/App.js* file.