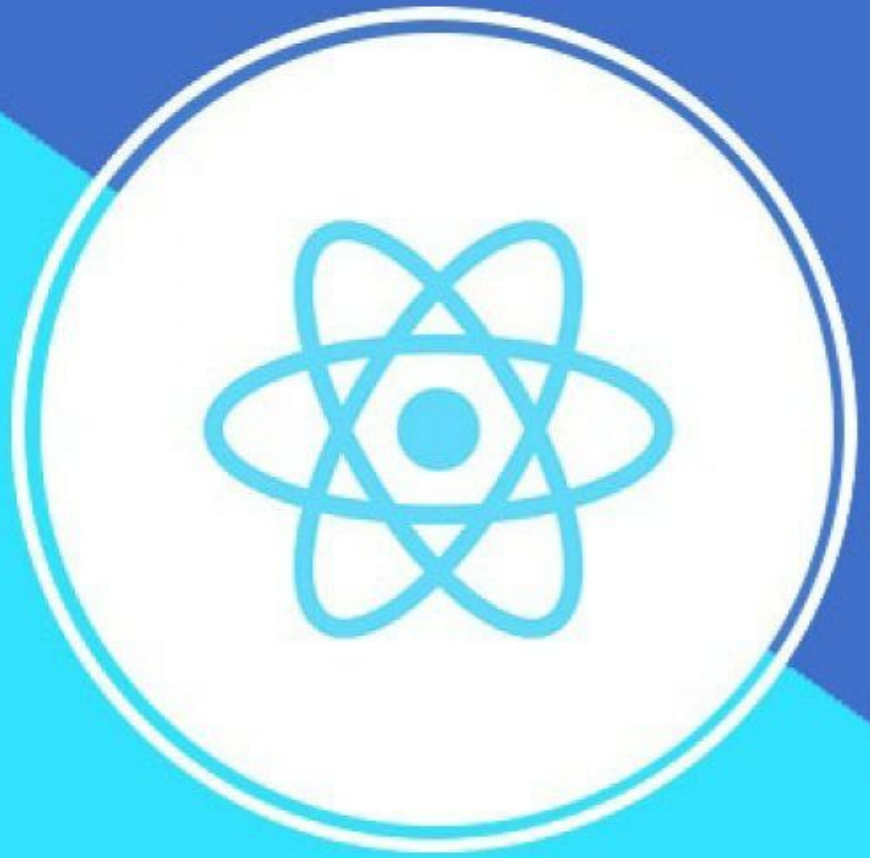

React Best Practices

1. Break larger components into smaller ones

To make components reusable, break the larger component into smaller ones. That means instead of putting multiple functionalities in one component create one component that will be responsible for single functionality, the principle is called as "single responsibility principle".

Separating components will help us to read, maintain, debug, reuse and scale code in long run, as the project keeps on getting bigger and bigger with time due to the addition of new features.

2. Reusability is important, so keep creation of new components to the minimum required

By sticking to the rule of one function = one component, you can improve the reusability of components. What this means is that you should skip trying to build a new component for a function if there already exists a component for that function.

By reusing components across your project or across any number of projects, not only will you achieve consistency, you'll also be contributing to the community.

2. Reusability is important, so keep creation of new components to the minimum required

For example, you can even go further and create a Button component that can handle icons. Then, each time you need a button, you'll have a component to use.

```
class IconButton extends React.Component {
  [...]
  render() {
    return (
      <button onClick={this.props.onClick()}>
        <i class={this.props.iconClass}></i>
      </button>
    );
  }
}
```

3. Comment only where necessary

Attach comments to code only where necessary. This is not only in keeping with React best practices, it also serves two purposes at the same time:

- It'll keep code visually clutter free.
- You'll avoid a potential conflict between comment and code, if you happen to alter the code at some later point in time.

4. Name the component after the function

It's a good idea to name a component after the function that it executes so that it's easily recognizable.

For example, ProductTable – it conveys instantly what the component does. On the other hand, if you name the component based on the need for the code, it can confuse you at a future point of time.

Besides, naming a component after the function makes it more useful to the community as it's more likely to be discovered.

5. Consolidate duplicate code – DRY your code

A common rule for all code is to keep it as brief and concise as possible.

It's no different here too, since React best practices also instruct you to keep code brief and precise. One way to do this is to avoid duplication – Don't Repeat Yourself (DRY).
This relies heavily on the reusability principle in React. Let's say you want to add multiple buttons that contain icons, instead of adding the markup for each button, you can simply use the IconButton component that we shown above. You could even go further by mapping everything into an array.

```
const buttons = ['facebook', 'twitter', 'youtube'];

return (
  <div>
    {
      buttons.map( (button) => {
        return (
          <IconButton
            onClick={doStuff( button )}
            iconClass={button}
          />
        );
      } )
    }
  </div>
);
```

6. Put CSS in JavaScript 🎨

When you start working on a project, it is a common practice to keep all the CSS styles in a single SCSS file. The use of a global prefix prevents any potential name collisions. However, when your project scales up, this solution may not be feasible.

Here's an example of using EmotionJS in your project. EmotionJS can generate complete CSS files for your production. First, install EmotionJS using npm.

```
npm install --save @emotion/core
```

Next, you need to import EmotionJS in your application.


```
npm install --save @emotion/core
```


You can set the properties of an element as shown in the snippet below:

```
let Component = props => {<div css={{ border: '1px'}}{...props}/>)}
```

7. Use capitals for component names 

If, like most folks, you're using JSX (a JavaScript extension), the names of the components you create need to begin with uppercase letters. For instance, you'll name components as SelectButton instead of selectbutton, or Menu instead of menu. We do this so that JSX can identify them differently from default HTML tags.

In case JSX is not your language of choice, you can use lowercase letters. However, this may reduce the reusability of components beyond your project.

8. Mind the other naming conventions 

When working with React, you are generally using JSX (JavaScript Extension) files. Any component that you create for React should therefore be named in Pascal case, or upper camel case. This translates to names without spaces and the capitalizing the first letter of every word.

If you want to create a function that submits a form, you should name it SubmitForm in upper camel case, rather than submitForm, submit_form, or submit_form. Upper camel case is more commonly called Pascal case.

9. Separate stateful aspects from rendering 🎭

One of React best practices is to keep your stateful data-loading logic separate from your rendering stateless logic. It's better to have one stateful component to load data and another stateless component to display that data. This reduces the complexity of the components.

For example, your app is fetching some data on mount. What you want to do is manage the data in the main component and pass the complex render task to a sub-component as props.

```
import RenderTable from './RenderTable';

class Table extends Component {
  state = { loading: true };

  render() {
    const { loading, tableData } = this.state;
    return loading ? <Loading/> : <RenderTable data=
{tableData}/>;
  }

  componentDidMount() {
    fetchTableData().then( tableData => {
      this.setState( { loading: false, tableData } );
    } );
  }
}
```

10. Use tools like Bit 

One of React best practices that helps to organize all your React components is the use of tools like Bit.

These tools help to maintain and reuse code. Beyond that, it helps code to become discoverable, and promotes team collaboration in building components. Also, code can be synced across projects.

11. Use snippet libraries 

Code snippets help you to keep up with the best and most recent syntax. They also help to keep your code relatively bug free, so this is one of the React best practices that you should not miss out on.

There are many snippet libraries that you can use, like, ES7 React, Redux, JS Snippets, etc.

12. Write tests for all code ✍️

Testing ensures code integrity. Therefore it is good practice to create a Test directory within your component's directory to perform all the required tests and ensure that the addition of new code will not break the existing functionality.

You can perform,

- Unit Testing to check individual components of React application.
- Integration Testing to check if different pieces of models are working well together
- End to end testing to check the entire application flow.

13. All files related to any one component should be in a single folder 📁

Keep all files relating to any one component in a single folder, including styling files.

If there's any small component used by a particular component only, it makes sense to keep these smaller components all together within that component folder. The hierarchy will then be easy to understand – large components have their own folder and all their smaller parts are split into sub-folders.
This way, you can easily extract code to any other project or even modify the code whenever you want.