Components & Props | Cheat Sheet

Concepts in Focus

- Component
  - Properties (Props)
  - Component is Reusable
  - Component is Composable
- Third-party Packages
  - create-react-app
  - Pre-Configured tools

1. Component

A Component is a JS function that returns a JSX element.

```
const Welcome = () => <h1 className="message">Hello, User</h1>;
```

The component name should always start with a capital letter as react treats the components starting with lowercase letters as HTML elements.

```
<script type="text/babel">
  const Welcome = () => <h1 className="message">Hello, User</h1>;
  ReactDOM.render(<Welcome />, document.getElementById("root"));
</script>
```

We can call the function with self-closing tags as shown above

`<Welcome />` .

1.1 Properties (Props)

React allows us to pass information to a component using props.

1.1.1 Passing Props

We can pass props to any component as we declare attributes for any HTML element.

Syntax:

```
<Component propName1="propValue1" propName2="propValue2" />
```

```
const Welcome = () => <h1 className="message">Hello, User</h1>;

ReactDOM.render(
  <Welcome name="Rahul" greeting="Hello" />,
  document.getElementById("root")
);
```

1.1.2 Accessing Props

The components accept props as parameters and can be accessed directly.

Syntax:

```
const Component = (props) => {
  // We can access props here
};
```

```
const Welcome = (props) => {
  const { name, greeting } = props;
  return (
    <h1 className="message">
      {greeting}, {name}
    </h1>
  );
};

ReactDOM.render(
  <Welcome name="Rahul" greeting="Hello" />,
  document.getElementById("root")
);
```

1.2 Component is Reusable

A Component is a piece of reusable code that can be used in various parts of an application.

Example:

```
const Welcome = (props) => {
  const { name, greeting } = props;
  return (
    <h1 className="message">
      {greeting}, {name}
    </h1>
  );
};

ReactDOM.render(
  <div>
    <Welcome name="Rahul" greeting="Hello" />
    <Welcome name="Ram" greeting="Hi" />
  </div>,
  document.getElementById("root")
);
```

1.3 Component is Composable

We can include a component inside another component.

```
const Welcome = (props) => {
  const { name, greeting } = props;
  return (
    <h1 className="message">
      {greeting}, {name}
    </h1>
  );
};
const Greetings = () => (
  <div>
    <Welcome name="Rahul" greeting="Hello" />
    <Welcome name="Ram" greeting="Hi" />
  </div>
);

ReactDOM.render(<Greetings />, document.getElementById("root"));
```

2. Third-party Packages

Creating a real-world app involves lot of setup because a large number of components need to be organised.

Facebook has created a third-party package,

`create-react-app` , to generate a ready-made React application setup.

2.1 create-react-app

Installation Command:

```
npm install -g create-react-app
```

It installs

`create-react-app` globally in our environment.

2.1.1 Creating a React Application

```
create-react-app myapp --use-npm
```

2.1.2 React Application Folder Structure

- public/folder: Where we will keep assets like images, icons, videos etc
- src/folder: Where we will do the majority of our work. All of our React components will placed here.
- node_modules
- package-lock.json

node_modules:

This directory contains dependencies and sub-dependencies of packages used by the current react app, as specified by package.json.

package-lock.json:

This file contains the exact dependency tree installed in

`node_modules` . This provides a way to ensure every team member have the same version of dependencies and sub-dependencies.

The

`index.js` in the path `src/folder/` is a starting point to the application. `App.js` , `App.css` are imported in this file.

2.1.3 Starting a React Application

Run the below command from the React application directory.

```
npm start
```

You can view the application in the URL

`http://localhost:3000` in your browser.

Note

All the ES6 Modules should be named with `.js` extension.

2.2 Pre-Configured tools

The

`create-react-app` comes pre-configured with:

- **Live editing**: Allows React components to be live reloaded.
- **ESLint**: Analyzes source code to report programming errors, bugs, and syntax errors.
- **Prettier**: Enforces a consistent style for indentation, spacing, semicolons and quotes, etc.
- **Babel**: Compiles JSX into Regular JavaScript
- **Webpack**: Stitches together a group of modules into a single file (or group of files). This process is called Bundling.