# React Notes - All in One File

## 1. Setting Up a React Application

Use tools like create-react-app or Vite to set up a React project.

```bash  
npx create-react-app my-app  
cd my-app  
npm start  
```

### Folder Structure:

```  
my-app/  
├── public/  
├── src/  
│ ├── App.js  
│ ├── index.js  
│ ├── components/  
│ └── styles/  
├── package.json  
└── README.md  
```

## 2. Entry Point: index.js

This is the starting point of a React application. It renders the App component into the DOM using ReactDOM.render().

```javascript  
import React from 'react';  
import ReactDOM from 'react-dom';  
import App from './App';  
  
ReactDOM.render(  
 <React.StrictMode>  
 <App />  
 </React.StrictMode>,  
 document.getElementById('root')  
);  
```

## 3. Root Component: App.js

The App component is the root component of the application. It acts as a container for other components.

```javascript  
import React from 'react';  
import Header from './components/Header';  
import MainContent from './components/MainContent';  
import Footer from './components/Footer';  
  
function App() {  
 return (  
 <div>  
 <Header />  
 <MainContent />  
 <Footer />  
 </div>  
 );  
}  
  
export default App;  
```

## 4. Creating Components

### Functional Components

Simple JavaScript functions that return JSX.

```javascript  
function Header() {  
 return <h1>Welcome to My React App</h1>;  
}  
```

### Class Components

ES6 classes that extend React.Component (less common in modern React).

```javascript  
class Header extends React.Component {  
 render() {  
 return <h1>Welcome to My React App</h1>;  
 }  
}  
```

## 5. JSX (JavaScript XML)

JSX is a syntax extension for JavaScript that allows you to write HTML-like code in React.

\*\*Example:\*\*

```javascript  
function Greeting() {  
 return <h1>Hello, World!</h1>;  
}  
```

## 6. Props (Properties)

Props are used to pass data from a parent component to a child component.

\*\*Example:\*\*

```javascript  
function Greeting(props) {  
 return <h1>Hello, {props.name}!</h1>;  
}  
  
function App() {  
 return <Greeting name="Amit" />;  
}  
```

## 7. State

State is used to manage data that changes over time within a component. Use the `useState` hook in functional components:

```javascript  
import React, { useState } from 'react';  
  
function Counter() {  
 const [count, setCount] = useState(0);  
  
 return (  
 <div>  
 <p>Count: {count}</p>  
 <button onClick={() => setCount(count + 1)}>Increment</button>  
 </div>  
 );  
}  
```

## 8. Event Handling

React uses camelCase for event handlers (e.g., `onClick`, `onChange`).

\*\*Example:\*\*

```javascript  
function Button() {  
 const handleClick = () => {  
 alert('Button clicked!');  
 };  
  
 return <button onClick={handleClick}>Click Me</button>;  
}  
```

## 9. Component Lifecycle (Class Components)

Lifecycle methods are used in class components to perform actions at specific stages (e.g., mounting, updating, unmounting).

Common lifecycle methods:  
- componentDidMount(): Runs after the component is rendered.  
- componentDidUpdate(): Runs after the component is updated.  
- componentWillUnmount(): Runs before the component is removed from the DOM.

## 10. Hooks (Functional Components)

Hooks allow functional components to use state and lifecycle features.

Common hooks:  
- useState: Manages state.  
- useEffect: Performs side effects (e.g., fetching data, updating the DOM).

```javascript  
import React, { useState, useEffect } from 'react';  
  
function DataFetcher() {  
 const [data, setData] = useState([]);  
  
 useEffect(() => {  
 fetch('https://api.example.com/data')  
 .then((response) => response.json())  
 .then((data) => setData(data));  
 }, []);  
  
 return (  
 <ul>  
 {data.map((item) => (  
 <li key={item.id}>{item.name}</li>  
 ))}  
 </ul>  
 );  
}  
```

## 11. Rendering Lists

Use the map() function to render lists of data.

\*\*Example:\*\*

```javascript  
function TodoList() {  
 const todos = ['Learn React', 'Build a project', 'Deploy to production'];  
  
 return (  
 <ul>  
 {todos.map((todo, index) => (  
 <li key={index}>{todo}</li>  
 ))}  
 </ul>  
 );  
}  
```

## 12. Conditional Rendering

Use JavaScript conditional statements (if, &&, ternary operator) to render content conditionally.

\*\*Example:\*\*

```javascript  
function Greeting({ isLoggedIn }) {  
 return isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>;  
}  
```

## 13. Styling in React

### Inline Styles

Use the style attribute with a JavaScript object.

```javascript  
function StyledComponent() {  
 return <div style={{ color: 'red', fontSize: '20px' }}>Hello, World!</div>;  
}  
```

### CSS Modules

Import CSS files as modules.

```javascript  
import styles from './MyComponent.module.css';  
  
function MyComponent() {  
 return <div className={styles.myClass}>Hello, World!</div>;  
}  
```

## 14. React Router (For Navigation)

Use react-router-dom to handle routing in a React app.

\*\*Example:\*\*

```javascript  
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';  
import Home from './Home';  
import About from './About';  
  
function App() {  
 return (  
 <Router>  
 <Routes>  
 <Route path="/" element={<Home />} />  
 <Route path="/about" element={<About />} />  
 </Routes>  
 </Router>  
 );  
}  
```

## 15. State Management (Optional)

For larger applications, use state management libraries like Redux or Context API.

\*\*Example with Context API:\*\*

```javascript  
import React, { createContext, useContext } from 'react';  
  
const ThemeContext = createContext();  
  
function App() {  
 return (  
 <ThemeContext.Provider value="dark">  
 <Toolbar />  
 </ThemeContext.Provider>  
 );  
}  
  
function Toolbar() {  
 const theme = useContext(ThemeContext);  
 return <div>Current Theme: {theme}</div>;  
}  
```

## 16. Building and Deploying

Build the app for production:

```bash  
npm run build  
```

Deploy to platforms like Netlify, Vercel, or GitHub Pages.

## 🎯 Summary of React Flow

1. Setup: Create a React app using create-react-app or Vite.  
2. Entry Point: index.js renders the App component.  
3. Components: Break the UI into reusable components.  
4. Props: Pass data between components using props.  
5. State: Manage dynamic data using useState.  
6. Events: Handle user interactions with event handlers.  
7. Lifecycle: Use useEffect for side effects.  
8. Routing: Add navigation with react-router-dom.  
9. Styling: Use inline styles or CSS modules.  
10. State Management: Use Context API or Redux for global state.  
11. Build & Deploy: Build and deploy the app for production.