Comprehensive Documentation for React.js

Introduction to React.js

React.js is a JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications where UI components dynamically update without requiring a full page reload. It uses a declarative programming paradigm and follows a component-based architecture.

Single Page Applications (SPA)

A Single Page Application (SPA) is a web application that dynamically updates content without refreshing the entire page. SPAs improve user experience by using AJAX and client-side routing.

Understanding SPA Features

- **Fast Loading:** Only required data is fetched.
- **Improved Performance:** Reduces server load and speeds up page transitions.
- **Client-Side Routing:** Uses libraries like React Router for smooth navigation.
- **State Management:** Manages data efficiently using tools like Redux or Context API.

Node Package Manager (NPM)

NPM is a package manager for JavaScript that allows developers to install and manage dependencies required for a React application. It is included with Node.js.

Create New App Using Vite

Vite is a modern build tool that provides faster and optimized project setup compared to Create-React-App.

To create a new React project using Vite:

```
# Install Vite globally (optional)
npm create vite@latest my-app --template react

# Navigate into the project folder
cd my-app

# Install dependencies
npm install
```

```
# Start the development server
npm run dev
```

Understanding Project Folder Structure

- **node_modules/** - Contains installed dependencies.
- **public/** - Holds static files like index.html.
- **src/** - Contains source code.
- **package.json** - Configuration file for dependencies.
- **vite.config.js** - Configuration file for Vite.

NPX

NPX is a package runner for executing npm packages without globally installing them. It is useful for running Vite without installing it globally.

Introduction to JSX

JSX (JavaScript XML) is a syntax extension for JavaScript that allows writing HTML-like code inside JavaScript.

Transpiling JSX to JavaScript Using Babel

Babel is a JavaScript compiler that converts JSX into standard JavaScript that browsers can understand.

Working with JSX

JSX enables writing HTML elements in React components:

```
const element = <h1>Hello, World!</h1>;
```

JSX Restrictions

- Must return a single parent element.
- HTML attributes should be camelCase (e.g., `className` instead of `class`).
- Expressions inside JSX must be enclosed in `{}`.

React Fragment

A special wrapper to group elements without adding extra DOM nodes.

```
<React.Fragment>
  <h1>Title</h1>
  <p>Paragraph</p>
</React.Fragment>
```

Component Architecture

React applications are built using components that encapsulate UI logic and behavior.

Creating Components

Components can be created as functions or classes:

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

Rendering Components

Components are rendered inside the root element using ReactDOM.

```
ReactDOM.createRoot(document.getElementById('root')).render(<Welcome />);
```

Understanding Component Basics

Components accept inputs called `props` and maintain internal state.

Stateless Components and Stateful Components

- **Stateless Components:** Do not manage state.
- **Stateful Components:** Maintain and manage state.

Functional Components

Simpler components that receive `props` and return JSX.

```
const Greeting = (props) => <h1>Hello, {props.name}!</h1>;
```

Pure Components

Pure components optimize rendering by preventing unnecessary updates.

```
class MyComponent extends React.PureComponent {}
```

Applying Styles to Components

Styles can be applied using inline styles, CSS files, or styled-components.

```
const style = { color: 'blue' };
<h1 style={style}>Styled Text</h1>;
```

Higher Order Components (HOC)

A Higher Order Component (HOC) is a function that takes a component and returns a new component with additional functionalities.

Example of a HOC for Logging

```
import React from 'react';

// Higher Order Component
const withLogger = (WrappedComponent) => {
  return (props) => {
    console.log('Rendering component:', WrappedComponent.name);
    return <WrappedComponent {...props} />;
  };
};

// Regular component
const HelloWorld = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};

// Enhancing component using HOC
const EnhancedHelloWorld = withLogger(HelloWorld);

// Usage
const App = () => {
  return <EnhancedHelloWorld name="John" />;
};

export default App;
```

React Context API

Used to pass data deeply in the component tree without prop drilling.

```
const MyContext = React.createContext();
```

Error Boundaries

Special components that catch JavaScript errors in child components.

```
class ErrorBoundary extends React.Component {
  componentDidCatch(error, info) {
    console.log(error, info);
  }
  render() {
    return this.props.children;
```

```
  }
}
```

Lazy Loading

Lazy loading loads components only when needed to improve performance.

```
const LazyComponent = React.lazy(() => import('./LazyComponent'));
```

React Router DOM

React Router is a library for handling navigation in React applications. It enables client-side routing, ensuring faster navigation without full page reloads.

Installing React Router DOM

```
npm install react-router-dom
```

Rules for Creating Routes

1. Wrap the application with `BrowserRouter` in `main.jsx`:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

2. Define routes using `Routes` and `Route` components in `App.jsx`:

```
import { Routes, Route } from 'react-router-dom';
import Home from './Home';
import About from './About';

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
    </Routes>
  );
};

export default App;
```

3. Use `Link` or `NavLink` for navigation instead of anchor (`<a>`) tags:

```
import { Link } from 'react-router-dom';

const Navbar = () => {
  return (
    <nav>
      <Link to="/">Home</Link>
      <Link to="/about">About</Link>
    </nav>
  );
};
```

Exercises on React Router

1. Create a 404 Page

Modify the `App.jsx` file to handle unknown routes by adding a wildcard route (`*`).

Solution:

```
const NotFound = () => {
  return <h2>404 - Page Not Found</h2>;
};

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="*" element={<NotFound />} />
    </Routes>
  );
};
```

2. Redirect a Route

Redirect `/old-about` to `/about` using `Navigate`.

Solution:

```
import { Navigate } from 'react-router-dom';

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
```

```
      <Route path="/old-about" element={<Navigate to="/about" />} />
    </Routes>
  );
};
```

3. Create Dynamic Routing

Create a dynamic route for user profiles (`/user/:id`).

Solution:

```
import { useParams } from 'react-router-dom';

const UserProfile = () => {
  const { id } = useParams();
  return <h2>User Profile for ID: {id}</h2>;
};

const App = () => {
  return (
    <Routes>
      <Route path="/user/:id" element={<UserProfile />} />
    </Routes>
  );
};
```

Understanding & Using State

Solutions to Exercises

1. **Toggle Visibility**

```
import { useState } from 'react';

const ToggleText = () => {
  const [isVisible, setIsVisible] = useState(true);

  return (
    <div>
      <button onClick={() => setIsVisible(!isVisible)}>Toggle Text</button>
      {isVisible && <p>This is a toggled message.</p>}
    </div>
  );
};

export default ToggleText;
```

2. **Counter with Step Increment**

```jsx
import { useState } from 'react';

const StepCounter = () => {
  const [count, setCount] = useState(0);
  const [step, setStep] = useState(1);

  return (
    <div>
      <input type="number" value={step} onChange={(e) => setStep(Number(e.target.value))} />
      <button onClick={() => setCount(count + step)}>Increment</button>
      <p>Current Count: {count}</p>
    </div>
  );
};

export default StepCounter;
```

3. **Dynamic Background Color**

```jsx
import { useState } from 'react';

const colors = ['red', 'blue', 'green', 'yellow', 'purple'];

const ColorChanger = () => {
  const [index, setIndex] = useState(0);

  return (
    <div style={{ backgroundColor: colors[index], height: '100vh', padding: '20px' }}>
      <button onClick={() => setIndex((index + 1) % colors.length)}>Change Color</button>
    </div>
  );
};

export default ColorChanger;
```