React Basics

React is a popular JavaScript library for building user interfaces, particularly single-page applications where you need to manage the view layer for web and mobile apps. It allows you to create reusable UI components, which can manage their own state. Here's a guide to getting started with React, including setting up a project, understanding JSX, components, and the component lifecycle.

1. Setting Up a React Project

a. Using Create React App

The easiest way to set up a new React project is by using the `Create React App` tool, which sets up everything you need to start developing a React application.

Step 1: Install Node.js Before you start, make sure you have Node.js installed. You can download it from the official website.

Step 2: Create a New React App Open your terminal and run the following command:

bash

```
npx create-react-app my-app
```

- `npx` comes with Node.js (version 5.2 and above) and runs the `create-react-app` command without installing it globally.
- `my-app` is the name of the directory where your app will be created.

Step 3: Start the Development Server Navigate to your project directory and start the development server:

bash

```
cd my-app
npm start
```

This command will open your new React app in a web browser, typically at `http://localhost:3000/`.

2. Understanding JSX

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML directly within JavaScript. It's an essential part of writing React components.

Example of JSX:

jsx

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

- JSX allows you to write HTML-like syntax directly in your JavaScript code.
- It is not a string or HTML; it is syntax transformed into JavaScript calls by tools like Babel.

JSX Example in a React Component:

jsx

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Hello, world!</h1>
    </div>
  );
}

export default App;
```

- In the example above, the `App` component returns JSX. When rendered, it displays "Hello, world!" on the webpage.

3. Components in React

Components are the building blocks of a React application. A component in React is a JavaScript function or class that optionally accepts inputs (props) and returns a React element that describes how a section of the UI should appear.

a. Functional Components

Functional components are simple functions that return JSX.

jsx

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- `Welcome` is a functional component that takes `props` as an argument and returns a React element.

b. Class Components

Class components are more feature-rich. They are ES6 classes that extend `React.Component` and must have a `render()` method returning JSX.

jsx

```
import React, { Component } from 'react';

class Welcome extends Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

export default Welcome;
```

c. Props

Props (short for "properties") are inputs to components. They are passed to components via attributes and are used to pass data from one component to another.

Using Props:

jsx

```
function App() {
  return (
    <div>
      <Welcome name="Alice" />
      <Welcome name="Bob" />
    </div>
  );
}

function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- In this example, the `App` component renders the `Welcome` component twice, passing different names via props.

4. State in React

State is a built-in object that stores property values that belong to the component. When the state object changes, the component re-renders.

a. Using State in Functional Components with Hooks

React introduced hooks, like `useState`, to use state in functional components.

jsx

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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
```

```
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

- `useState` is a hook that allows you to add React state to functional components. It returns a pair: the current state value and a function to update it.

b. Using State in Class Components

State in class components is managed with `this.state` and updated with `this.setState()`.

jsx

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count:
this.state.count + 1 })}>
          Click me
        </button>
```

```
      </div>
    );
  }
}

export default Counter;
```

5. Component Lifecycle in React

Class components in React have a lifecycle, which consists of methods that get called at different stages of a component's life. The most common lifecycle methods are:

a. Mounting

Mounting refers to putting elements into the DOM. Key methods:

- **`constructor()`**: Initializes state and binds event handlers.
- **`componentDidMount()`**: Invoked immediately after the component is mounted. Perfect for making API calls.

b. Updating

Updating happens when a component’s state or props change.

- **`shouldComponentUpdate()`**: Determines whether a component should re-render.
- **`componentDidUpdate()`**: Called after a component has re-rendered.

c. Unmounting

Unmounting occurs when a component is removed from the DOM.

- **`componentWillUnmount()`**: Invoked before the component is unmounted and destroyed. Used for cleanup, such as removing timers or cancelling network requests.

Example of Component Lifecycle Methods:

jsx

```
import React, { Component } from 'react';
```

```
class LifeCycleDemo extends Component {
  constructor(props) {
    super(props);
    console.log('Constructor: Initializing state');
    this.state = { data: null };
  }

  componentDidMount() {
    console.log('ComponentDidMount: Fetching data');
    // Simulate data fetch
    setTimeout(() => {
      this.setState({ data: 'Hello, World!' });
    }, 1000);
  }

  componentDidUpdate(prevProps, prevState) {
    console.log('ComponentDidUpdate: Component was updated');
  }

  componentWillUnmount() {
    console.log('ComponentWillUnmount: Cleaning up');
  }

  render() {
    console.log('Render: Displaying component');
    return (
      <div>
        <h1>Lifecycle Demo</h1>
        <p>{this.state.data}</p>
      </div>
    );
  }
}
```

```
export default LifeCycleDemo;
```

- When you run this component, you can observe the lifecycle method calls in the console.

Summary

- **Setup**: Use `Create React App` to set up a new React project.
- **JSX**: JSX is a syntax extension that looks like HTML but is actually JavaScript.
- **Components**: Components are reusable UI elements in React. They can be functional or class-based.
- **State**: State represents data that can change over time and affects how the component renders.
- **Lifecycle**: Class components have lifecycle methods that allow you to execute code at specific points during a component's life.