React.js Guide

1. What is React.js?

React.js is a **JavaScript library** for building user interfaces, developed by Facebook. It's component-based, meaning UIs are broken down into reusable pieces called **components**.

Why React.js?

- **Component-Based**: Breaks down UI into small, reusable components.
- **Virtual DOM**: Efficiently updates and renders UI changes.
- **Unidirectional Data Flow**: Ensures data flows in one direction, making the app more predictable.
- **Popular and In-Demand**: Large community and vast ecosystem of tools and libraries.

---

2. React.js Fundamentals

JSX (JavaScript XML)

JSX allows you to write HTML-like code inside JavaScript, which React then converts into real HTML.

Example:

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

JSX can also handle expressions and dynamic content:

```
const name = 'Yatharth';
const element = <h1>Hello, {name}!</h1>;
```

Components

Components are the building blocks of a React app. There are two main types of components:

1. **Functional Components** (Simple, stateless)
2. **Class Components** (Used for managing state before React Hooks)

Functional Component Example:

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

Class Component Example:

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

Props (Properties)

Props are inputs to components. They allow data to flow from parent to child components.

Example:

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

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

---

3. State in React

State is used to store and manage data within a component. Unlike props, which are read-only, state can be modified within the component.

Using State in a Class Component

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

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
```

```
  }
}
```

Using State in a Functional Component (with Hooks)

With React Hooks, you can now manage state in functional components using `useState`.

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

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

---

4. React Hooks

React Hooks allow you to use state and other React features in functional components. Some commonly used hooks are:

useState

Manages local state within a functional component.

```
const [state, setState] = useState(initialState);
```

useEffect

Performs side effects (e.g., data fetching, setting up subscriptions).

```
useEffect(() => {
  // Effect logic
}, [dependencies]);
```

Example: Fetching data with `useEffect`

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

```
function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(result => setData(result));
  }, []);

  return <div>Data: {JSON.stringify(data)}</div>;
}
```

useContext

Manages global state by using React's Context API.

```
const MyContext = React.createContext();
function MyComponent() {
  const value = useContext(MyContext);
  return <div>{value}</div>;
}
```

---

5. Event Handling in React

Event handling in React is similar to HTML, but React uses camelCase for event names and passes events as functions.

Example of an Event Handler

```
function Button() {
  function handleClick() {
    alert('Button clicked!');
  }

  return <button onClick={handleClick}>Click me</button>;
}
```

---

6. Conditional Rendering

You can render different UI based on conditions, similar to JavaScript's `if` and `ternary` operators.

Example:

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign in.</h1>;
  }
}
```

Ternary Operator Example:

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

---

7. Lists and Keys

Rendering lists in React is done using the `.map()` method. Each list item should have a unique `key` prop.

Example: Rendering a List

```
function ItemList(props) {
  const items = props.items;
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

---

8. Forms in React

Forms in React require controlled components, where the input's value is controlled by React state.

Example: Controlled Form

```
function NameForm() {
  const [name, setName] = useState('');

  function handleSubmit(event) {
    event.preventDefault();
```

```
    alert('A name was submitted: ' + name);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)}
/>
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

---

9. Routing with React Router

React Router allows you to add navigation to your app. First, install React Router:

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

Basic Routing Setup:

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

function Home() {
  return <h2>Home</h2>;
}

function About() {
  return <h2>About</h2>;
}

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/about" component={About} />
      </Switch>
    </Router>
  );
}
```

---

10. State Management with Redux

Redux is a popular library for managing application-wide state in larger applications. It centralizes state in a **store** and updates it through **actions** and **reducers**.

Redux Setup Example

1. Install Redux:

```
npm install redux react-redux
```

2. **Create a Reducer**:

```
function counterReducer(state = 0, action) {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
    default:
      return state;
  }
}
```

3. **Create the Store**:

```
import { createStore } from 'redux';
const store = createStore(counterReducer);
```

4. **Dispatch Actions**:

```
store.dispatch({ type: 'INCREMENT' });
console.log(store.getState());  // 1
```

---

11. Testing React Components

React Testing Library and Jest are commonly used for testing React components.

Example of Testing a Component:

```
import { render, screen } from '@testing-library/react';
import '@testing-library/jest-dom';
import App from './App';
```

```
test('renders welcome message', () => {
  render(<App />);
  const linkElement = screen.getByText(/Welcome to React/i);
  expect(linkElement).toBeInTheDocument();
});
```

---

12. Learning Resources

- **Official Documentation**: React Docs