React.js Full Theory Crash Course (Beginner to Advanced)

PART 1: FUNDAMENTALS

...[Part 1 and 2 content remains unchanged]...

PART 3: LIFECYCLE METHODS & CLASS COMPONENT DETAILS

...[Part 3 content remains unchanged]...

PART 4: REACT ROUTER AND NAVIGATION

1. What is React Router?

React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, using browser URL as the state.

2. Installation

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

3. Core Concepts

- **BrowserRouter**: Wraps your app and enables routing using the browser's history.
- **Routes and Route**: Define individual routes and which component they render.
- **Link**: Used to navigate without reloading the page.
- **useNavigate**: Programmatic navigation.

4. Example

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
```

```
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

5. Nested Routes

You can render components inside other components using nested routing.

6. Dynamic Routes

```
<Route path="/user/:id" element={<UserProfile />} />
```

Use `useParams()` to extract URL parameters.

7. Redirects

Use `<Navigate to="/login" />` for programmatic redirects.

PART 5: ADVANCED PATTERNS

1. Higher-Order Components (HOC)

A function that takes a component and returns a new component.

```
function withLogger(Component) {
  return function WrappedComponent(props) {
    console.log('Rendering', Component.name);
    return <Component {...props} />;
  };
}
```

2. Render Props

A technique for sharing code between React components using a function prop.

```
<DataProvider render={data => <SomeComponent data={data} />} />
```

3. Compound Components

Group related components together for better reusability and encapsulation.

4. Controlled vs Uncontrolled Components

- Controlled: Form input is tied to component state.
- Uncontrolled: DOM handles input state directly using refs.

PART 6: PERFORMANCE OPTIMIZATION

1. React.memo

Prevents unnecessary re-renders for functional components.

```
const MemoizedComponent = React.memo(MyComponent);
```

2. useMemo & useCallback

Memoize values and functions to avoid recalculating on every render.

3. Lazy Loading

Split code using `React.lazy` and `Suspense`.

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

4. Key Optimization Tips

- Avoid anonymous functions inside render
- Minimize state where not required
- Batch state updates
- Use pagination/virtual scroll for long lists

PART 7: TESTING IN REACT

1. Tools

- **Jest**: Testing framework for JavaScript.
- **React Testing Library**: Lightweight testing utility.

2. Basic Test Example

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

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

3. Mocking

Use `jest.fn()` or `jest.mock()` to mock functions or modules.

4. Coverage

Run `npm test -- --coverage` to check how much of your code is covered by tests.

PART 8: STATE MANAGEMENT

1. Local State

Managed via `useState` or `useReducer` in a single component.

2. Context API

Provides global state across the component tree.

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

3. Redux (optional)

Popular third-party library for managing global state.

- **Store**: Centralized state container
- **Actions**: Plain JS objects describing changes
- **Reducers**: Pure functions returning new state
- **Dispatch**: Sends action to reducer

4. Zustand, Recoil

Modern alternatives to Redux with simpler APIs.

PART 9: DEPLOYMENT & TOOLING

1. Build Tools

- **Vite** and **Webpack** are common tools to bundle React apps.

- `npm run build` creates a production build.

2. Hosting Platforms

- Vercel
- Netlify
- GitHub Pages
- Firebase Hosting

3. Environment Variables

Use `.env` files with variables prefixed by `REACT_APP_`.

4. ESLint & Prettier

Ensure code quality and consistent formatting.

✅You've completed the full theory crash course for React.js. You're now ready for interviews, advanced development, and project architecture.

Let me know if you'd like a downloadable **PDF** version!