React JS Notes

Component Basics

- **Component**: Reusable UI unit; can be a function or class.
- **JSX**: Syntax extension that looks like HTML; compiled to `React.createElement`.
- **Props**: Read only inputs passed from parent to child (`function MyComponent({title})`).
- **State**: Mutable data managed within a component (`useState`, `this.state`).

Functional vs. Class Components

- **Functional**: Preferred; use hooks for state & side effects.
- **Class**: `extends React.Component`; uses `render()`, `state`, lifecycle methods.

Hooks (Functional Only)

- `useState(initialValue)` !’ returns `[value, setValue]`.
- `useEffect(effect, deps?)` !’ runs side effects; cleanup via return function.
- `useContext(Context)` !’ accesses nearest Provider value.
- `useReducer(reducer, initState)` !’ Redux like state management.
- `useMemo(() => compute, deps)` !’ memoizes expensive calculations.

- `useCallback(() => fn, deps)` !’ memoizes functions to prevent re creation.
- `useRef(initial)` !’ mutable ref object (`ref.current`).

Lifecycle (Class) / Effect (Function)

Phase	Class Method	Hook Equivalent
Mount	`constructor`	`useEffect(..., [])`
	`componentDidMount`	
Update	`componentDidUpdate`	`useEffect(..., [deps])`
Unmount	`componentWillUnmount`	cleanup in `useEffect`
Error handling	`componentDidCatch`	`ErrorBoundary` (class)

Context API
---
1. `const MyContext = React.createContext(defaultValue);`
2. `<MyContext.Provider value={...}>` wraps part of tree.
3. Child: `const value = useContext(MyContext);`

Routing (react router)

- `<BrowserRouter>` at app root.
- `<Routes>` & `<Route path="..." element={<Comp/>}>`.
- Navigation: `useNavigate()` or `<Link to="/path">`.

State Management Options

- **Local**: `useState` / `useReducer`.
- **Context**: for global-ish data without prop drilling.
- **Redux / Zustand / Recoil**: external stores for large apps.

Performance Tips
- Keep component hierarchy shallow.
- Memoize pure components: `React.memo(Component)`.
- Use `useMemo`/`useCallback` to avoid unnecessary renders.
- Lazy load routes & heavy components: `React.lazy` + `<Suspense>`.
- Avoid anonymous functions/objects in JSX props.

Testing
- **Jest**: test runner & assertions.
- **React Testing Library**: render components, query by role/text, fire events.
- Snapshot testing with `react-test-renderer` (use sparingly).

Common Best Practices
- Prefer functional components & hooks.
- Keep components small & focused (single responsibility).
- Use TypeScript for type safety.

- Name files `ComponentName.jsx` / `ComponentName.tsx`.
- Export default component, named exports for helpers.
- Keep CSS scoped (CSS Modules, styled components, emotion).
- Validate props with `prop-types` (or rely on TypeScript).
- Use ESLint + Prettier for code consistency.

Useful Commands

- `npx create-react-app my-app` – scaffold project.
- `npm start` – dev server (hot reload).
- `npm run build` – production bundle.
- `npm test` – run tests.
- `npm i react-router-dom` – install router.
- `npm i @reduxjs/toolkit react-redux` – install Redux Toolkit.

Key Resources

- Official docs: https://reactjs.org/
- Hooks reference: https://reactjs.org/docs/hooks-reference.html
- React Router docs: https://reactrouter.com/
- Redux Toolkit docs: https://redux-toolkit.js.org/

End of React JS notes.