12-Day React Training Program

Goal: By the end of 9 days, trainees can build production-ready React apps with proper architecture, hooks, performance optimization, and deployment.

Day 1 — React Basics & Setup

What to Learn (Why)

- What is React & why SPA?
- JSX & rendering basics.
- Components (function-based).
- Props vs state.

Task: Build a “Profile Card” component with props (name, role, avatar).

Day 2 — State & Events

What to Learn (Why)

- useState hook.
- Handling events (onClick, onChange).
- Conditional rendering.
- Lists & keys.

Task: Build a “Todo List” app (add/remove tasks).

Day 3 — Forms & Controlled Components

What to Learn (Why)

- Controlled vs uncontrolled inputs.
- Form validation basics.
- Two-way binding.

Task: Build a “Signup Form” with validation (name, email, password).

Day 4 — useEffect & Lifecycle

What to Learn (Why)

- useEffect hook.
- Cleanup functions.
- Fetching API data.

Task: Build a “GitHub User Finder” (search + fetch API).

Day 5 — Routing & Navigation

What to Learn (Why)

- React Router basics (v6).
- Nested routes.
- Dynamic routes & params.
- Navigation guards (auth basics).

Task: Add routing to the GitHub Finder (Home, Profile, Not Found page).

Day 6 — Context API & Advanced Hooks

What to Learn (Why)

- Context API (global state).
- useReducer.
- Custom hooks.

Task: Build a “Theme Switcher” (light/dark) using Context + useReducer.

Day 7 — Performance & Optimization

What to Learn (Why)

- React.memo, useMemo, useCallback.
- Lazy loading components.
- React DevTools.

Task: Optimize GitHub Finder with memoization + lazy loading.

Day 8 — Real-World Features & Testing

What to Learn (Why)

- Error boundaries.
- Forms with Formik/Yup.
- Testing basics (React Testing Library).

Task: Add error handling + form validation tests to Signup Form.

Day 9 — Advanced Practices & Deployment (NEW)

What to Learn (Why)

- Project architecture & folder structure.
- Security basics (XSS prevention).
- Code quality: ESLint + Prettier.
- Git hooks (Husky, lint-staged).
- Environment configs.
- Deployment (Vercel/Netlify).

Task (Capstone): Build & Deploy a “Mini Blog App”

- Features: Login, Protected Routes, CRUD posts.
- Must include:
    - Components & props
    - useState, useEffect, Context
    - Routing
    - Performance optimization
    - Error boundaries
    - Deployment

Redux State Management Program

Day 10 — Redux Basics & Store Setup

What to Learn (Why)

- What is Redux & why use it?
- Core concepts: Store, Actions, Reducers, Dispatch.
- Connecting React components with the Redux store.
- Setting up the Redux Provider in the app.

Task:
Build a simple **Counter App** where:

- Increment & Decrement actions are dispatched via buttons.
- Counter value is stored and managed via Redux state.

Deliverable:
A working counter app fully using Redux for state management.

Day 11 — Advanced Redux Patterns

What to Learn (Why)

- Slice-based state management using Redux Toolkit.
- Async thunks for handling API calls.
- Using `useSelector` to read state and `useDispatch` to dispatch actions.
- Best practices for organizing slices and state structure.

Task:
Build a **User Data Fetching Feature**:

- Create a slice for user state.
- Use an async thunk to fetch user data from a mock API.
- Display user info in a profile component.

Deliverable:
A user profile page fully powered by Redux with async data fetching.

Day 12 — Middleware, DevTools & Performance

What to Learn (Why)

- Custom Redux Middleware (e.g., logging actions).
- Integration of Redux DevTools for easier debugging.
- Immutability principles in state updates.
- Performance optimization tips in Redux.

Task: Build a **Todo app** with Redux that handles adding, deleting, and toggling todo items. Add asynchronous action using `redux-thunk` for fetching todos from a remote API.

Task: (Optional)

- Add a **Logging Middleware** to print every dispatched action.
- Ensure Redux DevTools are properly configured and used.
- Optimize state updates to prevent unnecessary re-renders.

Deliverable:
A fully optimized Redux setup with middleware logging, DevTools visibility, and good state practices.

Final Deliverable of Redux Module

A fully functional **Redux-integrated React application** where:

- Component state is managed centrally.
- Async API calls are handled with thunks.
- Middleware logs dispatched actions.
- State is properly immutable and optimized.
- Redux DevTools enabled for development.

Essential NPM Packages for React (compact cheat sheet)

> Quick note: `npm i <pkg>` installs a runtime dependency; add `-D` for dev tools (linters, formatters). Use `npm i -S` only if you prefer explicit flags.

Core / Setup

- **react**, **react-dom** — React runtime. (already implied)
- **vite** — fast dev server / bundler (preferred over CRA).
  `npm i -D vite`
  Introduce: Day 1 (project setup)

Routing

- **react-router-dom** — routing (v6).
  `npm i react-router-dom`
  Introduce: Day 5 (routing & navigation)

Data fetching / server state

- **axios** — promise based HTTP client (easy & popular).
  `npm i axios`
- **@tanstack/react-query** (React Query) — caching, background fetch, mutations (recommended for real apps).
  `npm i @tanstack/react-query`
  Introduce: Day 6 (fetching/APIs)

Forms & Validation

- **react-hook-form** — performant, simple forms with minimal re-renders.
  `npm i react-hook-form`
- **formik** (alternative) — full-featured forms.
  `npm i formik`
- **yup** — schema validation (works with both).
  `npm i yup`
  Introduce: Day 3 (forms); deeper on Day 8 (testing/validation)

State management

- **reduxjs/toolkit** — modern Redux (recommended when app is complex).
  `npm i @reduxjs/toolkit react-redux`
- **zustand** — lightweight alternative for simpler global state.
  `npm i zustand`
  Introduce: Day 6 (context/state patterns) — show both and tradeoffs

---

Styling & UI

- **tailwindcss** — utility-first CSS framework.
  `npm i -D tailwindcss postcss autoprefixer` (then init)
- **styled-components** / **@emotion/react** — CSS-in-JS libs.
  `npm i styled-components`
- **@mui/material @emotion/react @emotion/styled** — Material UI component library.
  `npm i @mui/material @emotion/react @emotion/styled`
- **bootstrap** (useful if teaching Bootstrap)
  `npm i bootstrap`
  Introduce: Day 7 (styling & UI libraries)

Component Utilities

- **clsx** — conditional className helper.
  `npm i clsx`
- **react-icons** or **lucide-react** — icon sets.
  `npm i react-icons`
  Introduce: Day 7 (UI polish)

Animations

- **framer-motion** — simple, high-quality animations.
  `npm i framer-motion`
  Introduce: Day 7 (optional UI polish)

Testing

- **@testing-library/react** — component testing best practice.
  `npm i -D @testing-library/react @testing-library/jest-dom`
- **jest** (if not using Vite+Vitest) or **vitest** with Vite — test runner.
  `npm i -D vitest`
- **msw** (Mock Service Worker) — mock API for tests/dev.
  `npm i msw`
  Introduce: Day 8 (testing) — run simple unit & integration tests

---

Dev Tools / Quality

- **eslint** + **@typescript-eslint/parser** + **plugin** — linting.
  `npm i -D eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin`
- **prettier** — formatting.
  `npm i -D prettier`
- **husky** + **lint-staged** — precommit hooks.
  `npm i -D husky lint-staged`
- **source-map-explorer** / **rollup-plugin-visualizer** — bundle analysis (optional).
  Introduce: Day 9 (code quality & deployment)

Dev Experience & TypeScript

- **typescript** + `@types/*` — static types.
  `npm i -D typescript` ; `npm i -D @types/react @types/react-dom`
  Introduce: Day 1 (if using TS) and throughout

API Caching / Offline

- **swr** (Vercel) — alternative to React Query (simple caching).
  `npm i swr`
  Introduce: Day 6 (alternatives to React Query)

Internationalization & Dates

- **react-i18next** — i18n in React.
  `npm i react-i18next i18next`

- **date-fns** — modern date utilities.
  `npm i date-fns`
  Introduce: Day 7 (polish features)

Charts & Data Viz

- **recharts** or **chart.js + react-chartjs-2** — charts.
  `npm i recharts`
  Introduce: Capstone as enhancement

Auth / Security helpers

- **js-cookie** — cookies handling (for auth tokens).
  `npm i js-cookie`

- **jwt-decode** — decode JWT on client.
  `npm i jwt-decode`
  Introduce: Day 9 (auth & protected routes discussion)

Misc / Useful

- **localforage** — better localStorage / IndexedDB.
  `npm i localforage`

- **react-query-devtools** — debug React Query caches.
  `npm i @tanstack/react-query-devtools`
- **react-helmet-async** — manage document head (SEO).
  `npm i react-helmet-async`
  Introduce: Day 3/Day 9 (SEO & head management)

Backend / Integration (demo)

- **json-server** — quick fake REST API for teaching.
  `npm i -D json-server`
  Introduce: Day 4 (API demo), Day 6 (fetching)

Frontend Libraries

1 date-fns / moment

- **Purpose:** For parsing, formatting, and manipulating dates.

Install:

```
npm install date-fns
# or
npm install moment
```

Usage:

```
import { format } from "date-fns";
console.log(format(new Date(), "dd-MM-yyyy"));
```

- **Used for:** Formatting timestamps, scheduling, reports.

---

2 formik

- **Purpose:** Simplifies form handling in React (state + validation).

Install:

```
npm install formik
```

Usage:

```
import { useFormik } from "formik";
const formik = useFormik({ initialValues: { name: "" }, onSubmit: v =>
console.log(v) });
```

- **Used for:** Controlled form management.

---

3 yup

- **Purpose:** Schema validation (often paired with Formik).

Install:

```
npm install yup
```

Usage:

```
import * as Yup from "yup";
const schema = Yup.object({ email: Yup.string().email().required() });
```

- **Used for:** Form validation.

---

4 classnames

- **Purpose:** Conditionally apply CSS classes.

Install:

```
npm install classnames
```

Usage:

```
import cn from "classnames";
<div className={cn("btn", { active: true })}>Click</div>;
```

- **Used for:** Dynamic styling.

5 uuid

- **Purpose:** Generate unique IDs (for keys, users, etc.)

Install:

```
npm install uuid
```

Usage:

```
import { v4 as uuidv4 } from "uuid";
console.log(uuidv4());
```

- **Used for:** Generating unique identifiers.

6 react-toastify

- **Purpose:** Display notifications & alerts in React.

Install:

```
npm install react-toastify
```

Usage:

```
import { toast } from "react-toastify";
toast.success("Success!");
```

Used for: Toast messages, success/error alerts.