

React.js Development

Course Duration ~75 Hours · 10 Weeks · Live Online · Build-Along · Deployed Portfolio Projects

Curriculum reviewed: July 2026 — the React ecosystem moves fast; this syllabus tracks the current model

TypeScript

Server Components

React Compiler Era

TanStack Query

Testing

Accessibility

In short: Build React the way it's written in 2026 — server-first, typed, tested, and fast without hand-tuning every render. Most React tutorials still teach the 2020 model: client-only apps, effect-based fetching, manual memoization. This course teaches the current one. Performance is taught for the React Compiler era — automatic memoization at build time, what the compiler can and can't do, and when manual intervention is still warranted (judgment, not "memoization is dead"). Server-first React is the baseline: Server Components, the "use client" boundary, Server Actions, streaming and Suspense — a standard interview topic — with Next.js App Router for production apps and Vite for pure SPAs. `useEffect` is treated as a last resort, with the modern alternatives taught explicitly. And the AI question is answered honestly: AI generates React quickly and generates it wrong in ways you have to be able to see — the value has shifted to architecture, review, debugging, performance and accessibility, and one project is exactly that review exercise. Built for fresh IT graduates, working developers, and the large group trained on 2020-era React who need to modernise.

Duration	10 weeks · ~75 hours · live + hands-on	Format	Live online · build-along · deployed portfolio projects
Prerequisite	Solid JavaScript fundamentals and basic HTML/CSS. TypeScript taught from first principles. No prior React experience required.	Curriculum	10 modules, from TypeScript foundations to production and capstone
Portfolio	7 deployed, production-style projects plus a defended capstone	Ideal for	Fresh IT graduates, working developers, and React developers modernising off 2020-era patterns

Why Learn React When AI Writes Components?

AI generates React quickly and generates it wrong in ways you have to be able to see. The value has shifted from typing components to architecture, review, debugging, performance and accessibility — judgment that AI can't supply and employers now screen for hard. This course treats AI tooling as an accelerator you review: you'll use it, learn the failure patterns it repeats, and complete a code-review project on AI-generated React that passes a casual glance.

Course Highlights

- Every module leads with what you build and ship, not what you "learn about"
- Compiler-era performance: automatic memoization, and the judgment for when to intervene
- Server-first from the start: Server Components, Server Actions, streaming, the client boundary
- `useEffect` as a last resort — modern data fetching with TanStack Query and Suspense
- TypeScript throughout — typed React is the default, not an upgrade
- Accessibility as a build requirement, not a checklist at the end
- Testing depth: Vitest, React Testing Library, Playwright, CI gates
- AI tooling used honestly — as an accelerator you review, never a substitute for understanding

Hands-on Projects & Portfolio

- **Typed Component Library** — accessible primitives, documented for other developers to use
- **Server-First Dashboard** — Server Components, streaming and Server Actions
- **Data-Heavy Interface** — TanStack Query: caching, optimistic updates and offline behaviour
- **Complex Multi-Step Form** — validation, server-side handling and proper pending states
- **Performance Rescue** — profile a deliberately slow app and fix it, with before/after evidence

- **Fully Tested Feature** — unit, integration and end-to-end coverage in CI
- **AI Code Review Exercise** — find and fix the bugs in AI-generated React that pass a casual glance
- **Capstone** — production application, deployed, monitored and defended

Tools & Stack

React 19 · TypeScript · Next.js · Vite · React Compiler · TanStack Query · React Hook Form · Zod · Tailwind CSS · Radix / shadcn-style primitives · Vitest · React Testing Library · Playwright · ESLint · Storybook · Node.js · Git · GitHub Actions · Vercel · Docker

Career Outcomes

React remains the dominant frontend library and the most commonly required frontend skill in job descriptions — but the bar has moved. Client-only React is now the baseline expectation; what differentiates candidates is server-first architecture, TypeScript, testing depth, performance and accessibility. As organisations consolidate into platform and design-system teams, developers who can build reusable, accessible components for other teams to consume are disproportionately valuable. This course targets:

- Frontend Developer · React Developer · Full-Stack Developer (React + Node) · UI Engineer
- Frontend Engineer in a platform or design-system team · Product Engineer · Mobile developer via React Native as an onward path

Who This Course Is For

- Final-year students and fresh IT graduates targeting frontend or full-stack roles
- JavaScript developers moving into React properly rather than by copy-paste
- Backend and full-stack engineers who need frontend competence they can defend
- **React developers trained on 2020-era patterns** — class components, effect-based fetching, manual memoization — who need to modernise
- Angular and Vue developers switching ecosystems
- Developers using AI tooling who want to review and fix what it produces

Module 1: Modern JavaScript & TypeScript for React

7 Hours

YOU SHIP

- A typed utility and component toolkit — props, events, generics and discriminated unions for UI state — that every later project builds on

TOPICS

- The ES features React leans on; modules; async/await; immutability patterns
- TypeScript properly: typing props, state, events; generics in components; discriminated unions for UI state
- Why typed React is now the default rather than an upgrade

Assessment: Quiz + Lab

Module 2: React Fundamentals

7 Hours

YOU SHIP

- A working interactive app with composed components, controlled forms and list rendering — plus a written mental model of when and why React re-renders

TOPICS

- Components and JSX; props and composition; rendering and reconciliation
- Lists and keys; conditional rendering; forms and controlled inputs
- A mental model of when and why React re-renders

Assessment: Quiz + Lab

Module 3: Hooks & State, the Modern Way

8 Hours

YOU SHIP

- A refactor that removes needless effects from a tutorial-style codebase — derived state, event responses and fetching moved to where they belong — plus a custom-hook library

TOPICS

- useState, useReducer, useContext, useRef; custom hooks
- useEffect as a last resort: what belongs in an effect and what doesn't
- The anti-patterns that survive in old tutorials — and how to recognise them

Assessment: Quiz + Lab**Module 4: Data Fetching & Server State**

7 Hours

YOU SHIP

- A data-heavy interface with TanStack Query — caching, invalidation, optimistic updates, retries and offline behaviour — wrapped in proper loading and error boundaries

TOPICS

- Why server state isn't UI state; TanStack Query for caching, invalidation, optimistic updates and retries
- Suspense for data; loading and error boundaries
- The use() pattern for reading promises

Assessment: Quiz + Lab**Module 5: Server-First React**

9 Hours

YOU SHIP

- A server-first dashboard: Server Components with a deliberate client boundary, Server Actions handling forms without hand-rolled API routes, streaming with pending UI

TOPICS

- Server Components and the "use client" boundary — what runs where, and why
- Server Actions and form handling; useActionState and pending UI
- Streaming and partial rendering; the security and bundle-size reasons this model won

Assessment: Quiz + Lab**Module 6: Routing & Application Architecture**

7 Hours

YOU SHIP

- A feature-organised application with nested layouts, route-level data loading and a structure that survives growth — plus the same app scoped as a Vite SPA, with a written justification for each path

TOPICS

- The App Router model; layouts and nested routes; route-level data loading
- Project structure that survives growth; feature-based organisation; separating UI primitives from feature components
- The SPA path with Vite and a client router where a meta-framework isn't warranted

Assessment: Quiz + Lab**Module 7: Styling & Design Systems**

8 Hours

YOU SHIP

- A typed, accessible component library — semantic, keyboard-navigable, focus-managed, themed with design tokens — documented for other developers to consume

TOPICS

- Utility-first CSS with Tailwind; component libraries and headless primitives; design tokens; theming
- Responsive and container-query layouts
- Accessibility as a build requirement: semantics, focus management, keyboard navigation, ARIA where genuinely needed

Assessment: Quiz + Lab

Module 8: Performance in the Compiler Era**8 Hours****YOU SHIP**

- A performance rescue: profile a deliberately slow app with React DevTools and fix it with evidence — code splitting, virtualisation, asset strategy — documented before/after against Core Web Vitals

TOPICS

- What the React Compiler optimises automatically and what it can't; writing compiler-friendly code
- Code splitting and lazy loading; list virtualisation; image and asset strategy; bundle analysis
- Core Web Vitals; React DevTools profiling; diagnosing a slow app with evidence rather than superstition

Assessment: Quiz + Lab

Module 9: Testing & Quality**7 Hours****YOU SHIP**

- A fully tested feature — unit, integration and end-to-end coverage with Vitest, React Testing Library and Playwright — running as CI gates alongside lint and type checks

TOPICS

- Vitest and React Testing Library; testing behaviour rather than implementation; mocking network calls
- Integration tests; end-to-end testing with Playwright; accessibility testing
- ESLint and type-checking as CI gates; a realistic testing pyramid on a real project

Assessment: Quiz + Lab

Module 10: Production & Capstone**7 Hours****YOU SHIP**

- Your capstone: one production-grade application — designed, built, tested, deployed and monitored — with the architecture defended in review, plus the AI code-review exercise

TOPICS

- Environment configuration and secrets; authentication patterns; error tracking and monitoring; CI/CD; deployment targets and their trade-offs
- Code review practice; working alongside AI tooling without shipping its mistakes
- Capstone design, build, deploy and defense

Assessment: Capstone Project + Review