5-Day Next.js Training — Interactive Project-Based Syllabus.

Goal: In 5 days, trainees will build and deploy a production-ready **Next.js Fullstack App** covering routing, SSR/SSG, APIs, styling, security, performance, testing, and deployment. The project grows day by day, ensuring each new concept builds on the last.

Day 1 — Getting Started with Next.js & Pages

Why: Understand how Next.js differs from React (SSR/SSG, file-based routing).

Learn

- What is Next.js? (SSR vs CSR vs SSG vs ISR) — Docs.
- File-based routing (pages/ App).
- Static vs dynamic routes.
- Linking with `<Link>`.
- Project setup with TypeScript + ESLint + Prettier.

Task:

- Create a project `next-blog` with `npx create-next-app@latest`.
- Build **Home Page** with navigation to `/about`.
- Add a **Layout component** for header/footer.

Outcome: A basic multi-page Next.js app running locally.

Day 2 — Data Fetching & Dynamic Pages

Why: Next.js' killer feature is pre-rendering with data.

Learn

- `getStaticProps` (SSG) — Docs.
- `getServerSideProps` (SSR) — Docs.
- `getStaticPaths` (dynamic SSG).

- Consuming external APIs.
- Image Optimization with `<Image>`.

Task 🛠️:

- Fetch posts from JSONPlaceholder API.
- Show a **Posts list** on `/posts`.
- Create a dynamic route `/posts/[id]` to render each post.
- Optimize thumbnails with Next.js `<Image>`.

Outcome: Blog now fetches and renders dynamic posts with static generation.

Day 3 — API Routes, Styling & Auth Basics

Why: Next.js is fullstack — backend APIs + frontend in one project.

Learn

- API Routes (`pages/api/*`) — Docs.
- Client-side fetching with `useEffect` & SWR.
- Styling options: CSS Modules, Tailwind, styled-components.
- Basic Auth pattern (JWT or NextAuth overview).
- Environment variables in Next.js.

Task:

- Add API route `/api/comments` that returns mock comments.
- Fetch & display comments for each post.
- Style app with **Tailwind CSS**.
- Add a simple **Login form** (no real backend, mock validation).

Outcome: Blog now supports API routes, styling, and a mock login system.

Day 4 — Advanced Features & Deployment

Why: Prepare app for production.

Learn

- Middleware (auth guard demo) — Docs.

- API mutations (POST, PUT, DELETE with fetch).
- Incremental Static Regeneration (ISR).
- SEO with `<Head>`.
- Deployment on **Vercel** — Docs.

Task:

- Add CRUD for posts (mock API + forms).
- Protect `/dashboard` route with Middleware (redirect if not logged in).
- Add SEO meta tags with `<Head>`.
- Deploy to **Vercel**.

Outcome: Blog is now CRUD-enabled, protected, SEO-friendly, and live online.

Day 5 — Security, Performance, Accessibility & Testing (NEW)

Why: Real apps need quality, security, and reliability.

Learn

- **Security**:
  - Preventing XSS with `dompurify`.
  - Safe use of environment variables (`NEXT_PUBLIC_` vs server-only) — Docs.
  - Cookie vs localStorage for tokens.
- **Performance**:
  - Lighthouse audits & Web Vitals — Docs.
  - Bundle analysis with `next-bundle-analyzer`.
  - Image & font optimization.
- **Accessibility (a11y)**:
  - Semantic HTML, aria roles.
  - Testing with axe DevTools.
- **Testing**:
  - Unit test with Vitest + React Testing Library.
  - API route test with Supertest (optional).
  - E2E demo with Playwright (optional).
- **Project Structure**:
  - Using `/lib` for helpers.
  - Modularizing features.
  - Client vs server components (App Router preview).

Task:

- Run Lighthouse on the deployed blog, fix performance issues.
- Add accessibility attributes (aria labels) on navigation.
- Write a unit test for the `PostCard` component.
- Secure login form against XSS.

Capstone Outcome: A **Production-Ready Blog App** with:

- Core features (CRUD posts, auth, comments).
- Security & a11y considerations.
- Performance optimized & tested.
- Clean structure + deployed live.

Essential NPM Packages (Cheat Sheet)

- `next`, `react`, `react-dom` — core runtime (installed by create-next-app).
- `swr` — React Hooks for data fetching (client side).
- `tailwindcss postcss autoprefixer` — styling.
- `next-auth` (optional) — authentication.
- `axios` — API client.
- `eslint prettier husky lint-staged` — code quality.
- `dompurify` — sanitize HTML.
- `@axe-core/react` — accessibility testing.
- `vitest @testing-library/react @testing-library/jest-dom` — testing.
