SEPTEMBER 2025

Comprehensive Study Guideline

NEXT.JS

WHITE MATRIX

PREPARED BY:
WHITE MATRIX
TRAINING TEAM

👋 Hey folks, why Next.js?
You've already learned React — awesome! 🎉 But here's the deal: React gives you the engine, and Next.js gives you the whole car. 🚗

- ⚡ **Faster apps** → with server-side rendering (SSR) and static site generation (SSG).
- 🔎 **Better SEO** → Google can actually "see" your pages, not just a blank div.
- 📁 **Zero-config routing** → just add files/folders, and boom → routes are ready.
- 🔌 **Built-in APIs** → no need to spin up a separate backend just for small endpoints.
- 🖼️ **Optimised images & performance** → apps load quicker, users happier.
- 🚀 **Easy deployment on Vercel** → one click, and your app is live worldwide.

So React taught you how to build components, but Next.js teaches you how to ship real apps.

00 Before you start prerequisites & tools

- Comfortable with React (components, hooks, state, props, effects).
- Node.js (v16+ recommended) and npm/yarn/pnpm installed.
- Basic CLI, Git, and JSON familiarity.
- Optional: VS Code, Docker (for advanced testing), a GitHub/GitLab account.

```
# create project
npx create-next-app@latest my-next-app
# or with pnpm:
pnpm create next-app@latest my-next-app
cd my-next-app
npm run dev
# opens http://localhost:3000
```

01 Project setup & folder structure (Pages vs App)

Goal
Understand how Next organises pages and components.

Key concepts:

- `pages/` router (older, stable): `pages/index.js` , `pages/about.js` , `pages/posts/[id].js.`
- `app/` router (modern; Next 13+): nested layouts, server components, `app/page.js` , `app/layout.js` , `app/posts/[slug]/page.js`.
- You can use either — recommendation: learn `pages/` first (clear data fetching APIs), then learn `app/` for server components & layouts.

Example file (pages):

`pages/index.js`

```jsx
jsx


export default function Home() {
  return <h1>Welcome to Next.js — built on React</h1>;
}
```

Example app router (app/page.js):

```jsx
jsx


export default function HomePage(){
  return <main><h1>Welcome (app router)</h1></main>
}
```

Mini-task 1

- Create a project, add two pages (`/` , `/about`) and navigate.
- Deliverable: link + screenshot of both pages and `package.json.`

02 Routing, Link, dynamic routes, nested routes

Goal

Use file-based routing; implement dynamic routes and client navigation.

Examples:

- Static route: `pages/about.js` or `app/about/page.js.`
- Dynamic route: `pages/posts/[id].js` or `app/posts/[id]/page.js.`
- Link component:

```
jsx

import  Link  from 'next/link';
<Link href = "/posts/123">Open post</Link>
```

Dynamic page example :

`(pages/posts/[id].js):`

```
jsx

import { useRouter } from 'next/router';
export default function Post(){
 const { query } = useRouter();
 return <div>Post id: {query.id}</div>;
}
```

Mini-project 2

- Build a simple blog index and dynamic post pages (static content OK).
- Deliverable: repo with `pages/index.js` listing posts and `pages/posts/[id].js.`

03 Data fetching patterns (SSG, SSR, ISR); pages router

Goal

Learn when & how to fetch data server-side vs build-time.

- **getStaticProps** — static generation at build time (SSG).
- **getStaticPaths** — generate dynamic routes at build time.
- **getServerSideProps** — server-side rendering (SSR) on each request.
- **Incremental Static Regeneration (ISR)** — revalidate static pages.

```jsx
import  Link  from 'next/link';
<Link href = "/posts/123">Open post</Link>
```

Example :

`pages/posts/index.js` (SSG)

```jsx
export async function getStaticProps(){
  const res = await fetch('https://api.example.com/posts');
  const posts = await res.json();
  return { props: { posts } };
}
```

Example :

`pages/posts/index.js` (SSG + getStaticPaths)

```jsx
export async function getStaticPaths(){ /* fetch ids and return paths */ }
export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();
  return { props: { post } };
}
```

Example :

`pages/report.js` (SSR)

```jsx
}
export async function getServerSideProps(ctx) {
  const res = await fetch('https://api.example.com/report');
  const data = await res.json();
  return { props: { data } };
}
```

Mini-project 3

- Create a blog using SSG for posts index, SSG+getStaticPaths for posts and implement ISR (`revalidate: 60`).
- Deliverable: repo + explanation of why used SSG/SSR/ISR for each page.

04 App router data fetching (server components)

Goal

Learn new `app/` router patterns if using Next 13+.

- In app router, server components are async and can `await` `fetch()` directly:

```jsx
// pages/api/hello.js
export default function handler(req, res){
  res.status(200).json({ message: 'Hello from API route' });
}
```

- Use `use client` directive at top of a component to make it client component if using state/hooks.

Mini-task 4

Convert one page to `app/` server component and one interactive client component (`use client`) for a comment box.

05 API routes (serverless functions)

Goal

Create backend endpoints inside the Next app.

- pages/api/hello.js or in App Router app/api/hello/route.js (the file API changed with app router)
- Example pages API:

```
// pages/api/hello.js
export default function handler(req, res){
  res.status(200).json({ message: 'Hello from API route' });
}
```

Example app router (edge/route):

```
// app/api/hello/route.js
export async function GET(request){
  return new Response(JSON.stringify({message:'Hi'}), {status:200});
}
```

Use cases:

- Proxy requests to third-party APIs.
- Form submission handling.
- Prototype internal endpoints for SSR/SSG to call.

Mini-project 5

- PBuild a simple notes API (`GET /api/notes` , `POST /api/notes`) storing data in-memory (for learning) or use a small DB (MongoDB).
- Deliverable: endpoints + example fetch from frontend.

06 Styling and assets

Goal

Use CSS Modules, global CSS, Tailwind, and Next Image optimisation.

- Global CSS: `styles/globals.css` imported in `pages/_app.js` or `app/layout.js.`
- CSS Modules: `styles/Home.module.css` used via `import styles from './Home.module.css'.`
- Tailwind: install & configure (very common in industry).
- Images: `<Image src="/me.png" width={200} height={200} alt="me" />` — optimized and responsive.

Mini-task 6

- Add Tailwind, style the blog list, use Next `<Image>` for a hero image.
  Deliverable: screenshots + tailwind config.

07 Styling and assets

Goal

Build interactive client-side parts (search, forms, pagination) using client components.

- Client components must include `'use client'` at top when using state/hooks in app router.
- Typical client side features: search filter, optimistic UI updates, form validation.

Example client component:

```
'use client'
import { useState } from 'react';
export default function LikeButton(){
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(c=>c+1)}>Like {count}</button>
}
```

Mini-project 7

Add a search box that filters blog posts client-side; or a simple comment form that POSTs to `/api/comments.` .

08 Authentication basics

Goal

Understand simple auth patterns and protecting pages/routes.

Patterns:

- JWT + cookies (httpOnly) — for token-based APIs.
- NextAuth.js — popular solution for OAuth + passwordless.
- Session cookies + API routes for login/logout.

Simple flow:

1. POST credentials to `/api/login` → validate (mock or DB) → return set-cookie with secure session token.
2. Use `getServerSideProps` /server component to check cookie and redirect if not logged in.

Mini-project 8

Implement a basic auth flow (mock user store), protect `/dashboard` with `getServerSideProps` redirect if not authenticated.

Deliverable: login page + protected dashboard and README on security caveats.

09 Performance, caching and SEO

Goal

Learn optimizations & SEO friendly practices.

- **Performance:**
  - Use SSG where possible.
  - Use dynamic `import()` for heavy components.
  - Image optimization, prefetching links (`<Link>` prefetch).
- **Caching:**
  - For pages fetch use `revalidate` or `cache-control.`
  - For API routes set cache headers.
- **SEO:**
  - `next/head` (pages) or `export const metadata` (app router) to set title/meta tags.
  - Canonical tags, OpenGraph tags, structured data.

Mini-project 9

Profile page with meta tags, optimized images, and lazy loaded charts.

10 Performance, caching and SEO

Goal:

Deploy to production and set up basic CI.

- Preferred: Vercel (native Next.js support). Basic steps: connect GitHub repo → `git push` → Vercel builds & deploys.
- Alternatively: Docker + Node server, or deploy to Netlify for pages router.
- CI: set up GitHub Actions to build & run tests, and deploy to Vercel by pushing to main.

Mini-project 10

- Deploy the blog app to Vercel and share the public URL. Add a `README` with deploy instructions.

11 Capstone Project (End-to-end)

Title:

Learn optimizations & SEO friendly practices.

Features to implement:

1. Home & product list (SSG for list, ISR for updates).
2. Product detail pages (SSG + getStaticPaths / server component).
3. Cart (client side; localStorage or server-backed).
4. Admin API (Next API routes) to add/update products (mocked or small DB).
5. Authentication for admin/dashboard (simple).
6. Deployment to Vercel.

Deliverables:

- Git repo with clear README.
- Public deployed URL.
- Short demo video (2–3 minutes) showing features.
- Short doc describing why SSG/SSR used where they are.

Suggested Exercises & Assessments

Goal:

- Weekly mini deliverable (one page or feature).
- Midterm: build blog with SSG, ISR, and API routes (deliverables: code + deployed link).
- Final: capstone demo + public deployment + short report.

Grading rubric (example):

- Functionality & correctness: 40%
- Code quality & structure: 20%
- Use of Next features (SSG/SSR/ISR/API routes): 20%
- Deployment & documentation: 20%

Useful Commands & Tips

- Start dev server: `npm run dev`
- Build for production locally: `npm run build && npm start`
- Lint: `npm run lint` (if included)
- Use `next/link` for navigation, `next/image` for images.
- For local API testing, use Postman or curl: `curl -X POST http://localhost:3000/api/notes`
- Keep secrets in `.env.local` (`NEXT_PUBLIC_` for client-exposed).

Recommended resources (short list)

- Official Next.js docs (search: "Next.js docs") — canonical and up to date.
- Vercel docs — deployment & platform features.
- Tutorials by Vercel and by freeCodeCamp for practical examples.
- Tailwind docs (if using Tailwind).
- MDN for web fundamentals (HTTP, REST, caching).

Quick - Tips Troubleshooting

- Component not updating? Check whether it's server component vs client component (app router).
- 404 on dynamic routes? Ensure `getStaticPaths` returns correct `paths.`
- API route CORS or missing headers? Set proper headers or call from same origin during development.
- Build fails on production only? Check Node version and SSR-only APIs (window/document only on client).

Final Note

This path is intentionally hands-on — small, focused modules with incremental mini-projects. Encourage the students to pair-program, keep short weekly demos, and commit early to Git. If you want, I can:

- generate a starter repo (create-next-app base + sample pages + SAM template if needed), or
- produce detailed step-by-step notebooks for any one module (for example App Router server components or auth) with full runnable code.

Thank You