Next.js Notes Book

A concise guide to understanding and using Next.js for building modern web applications.

Table of Contents

Introduction to Next.js

Next.js is a React framework for building server-side rendered (SSR), static, and hybrid web applications. It simplifies React development with built-in features like routing, API routes, and optimization.

- **Key Benefits**:
  - Automatic server-side rendering and static site generation
  - File-based routing
  - Built-in API routes
  - Optimized performance with image optimization and code splitting
- **Use Case**: Ideal for SEO-friendly, scalable web applications like blogs, e-commerce, or dashboards.

Core Features

Server-Side Rendering (SSR)

Next.js renders pages on the server for faster initial page loads and better SEO.

Static Site Generation (SSG)

Pages can be pre-rendered at build time for maximum performance.

Incremental Static Regeneration (ISR)

Update static content after deployment without rebuilding the entire site.

Automatic Code Splitting

Only loads the JavaScript needed for each page, improving performance.

Pages and Routing

Next.js uses a file-based routing system where files in the `pages` directory become routes.

- **Example**:
  - `pages/index.js` → `/`
  - `pages/about.js` → `/about`
  - `pages/blog/[id].js` → `/blog/1` (dynamic route)

Dynamic Routes

Create dynamic routes using square brackets for file names.

```
// pages/blog/[id].js
export default function BlogPost({ id }) {
  return <h1>Blog Post: {id}</h1>;
}

export async function getServerSideProps({ params }) {
  return { props: { id: params.id } };
}
```

Data Fetching

Next.js provides three main methods for fetching data:

getStaticProps (SSG)

Fetches data at build time for static pages.

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

getServerSideProps (SSR)

Fetches data on each request for server-rendered pages.

getStaticPaths (Dynamic Routes)

Defines dynamic routes for SSG.

```
export async function getStaticPaths() {
  return {
```

```
    paths: [{ params: { id: '1' } }, { params: { id: '2' } }],
    fallback: false,
  };
}
```

API Routes

Next.js allows you to create API endpoints in the `pages/api` directory.

Example API Route

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

- **Access**: `GET /api/hello` returns `{ "message": "Hello from Next.js API!" }`.

Best Practices

1. **Optimize Images**: Use Next.js's `<Image>` component for automatic optimization.
2. **Use Dynamic Imports**: Load non-critical components dynamically to reduce bundle size.
3. **Leverage ISR**: Use Incremental Static Regeneration for frequently updated static pages.
4. **Keep API Routes Light**: Avoid heavy logic in API routes; use them for simple data handling.
5. **SEO Optimization**: Use `next/head` to manage meta tags for better search engine visibility.
6. **Type Safety**: Integrate TypeScript for better maintainability.

Example: Simple Blog Page

Below is an example of a blog page using SSG with Next.js and Tailwind CSS for styling.

```
// pages/index.js
import Head from 'next/head';
import Link from 'next/link';

export default function Blog({ posts }) {
  return (
    <div className="min-h-screen bg-gray-100 py-10">
      <Head>
        <title>Next.js Blog</title>
        <meta name="description" content="A simple Next.js blog" />
      </Head>
      <div className="max-w-2xl mx-auto">
```

```
        <h1 className="text-4xl font-bold mb-8 text-center">Blog</h1>
        <ul className="space-y-4">
          {posts.map(post => (
            <li key={post.id} className="p-4 bg-white rounded shadow">
              <Link href={`/blog/${post.id}`}>
                <a className="text-xl text-blue-600
hover:underline">{post.title}</a>
              </Link>
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
}

export async function getStaticProps() {
  // Mock data (replace with API call)
  const posts = [
    { id: '1', title: 'First Post' },
    { id: '2', title: 'Second Post' },
  ];
  return { props: { posts } };
}
```

Steps to Use

1. Create a new Next.js project: `npx create-next-app@latest my-app`.
2. Replace `pages/index.js` with the above code.
3. Install Tailwind CSS: Follow Next.js Tailwind guide.
4. Run `npm run dev` to view the app at `http://localhost:3000`.

Additional Resources

- **Official Docs**: nextjs.org
- **Tutorials**: Vercel's Next.js Learn, YouTube channels like Traversy Media
- **Community**: Next.js Discord, GitHub Discussions