Chapter 06: Chuyển đổi sang Next.js - SSR, SSG và File-based Routing

A. Mục tiêu

Sau bài học này, học viên sẽ:

- Hiểu **Next.js** là gì và những vấn đề nó giải quyết so với ứng dụng React thông thường.
- Phân biệt rõ ràng các triết lý rendering: **CSR**, **SSR**, và **SSG**.
- Nắm vững cơ chế **File-system based routing** của Next.js.
- Biết cách thiết lập dự án Next.js và xây dựng các trang cơ bản.

B. Nội dung lý thuyết

1. Tại sao cần Next.js? - The React Framework

React là một **thư viện** tập trung vào việc xây dựng giao diện người dùng. Tuy nhiên, để phát triển một ứng dụng hoàn chỉnh, bạn cần tự xử lý nhiều vấn đề như:

- **Routing**: Cần sử dụng thư viện như `react-router-dom`.
- **Data fetching**: Tự viết logic để gọi API hoặc tải dữ liệu.
- **Tối ưu hóa hiệu năng**: Tự xử lý code-splitting, lazy loading, hoặc SEO.
- **Server-side logic**: Không có sẵn cơ chế để render phía server hoặc tạo API.

Next.js là một **framework** xây dựng trên React, cung cấp các giải pháp tích hợp sẵn cho các vấn đề trên. Nó giúp:

- Tăng tốc phát triển ứng dụng với cấu trúc rõ ràng.
- Tối ưu hóa hiệu năng và SEO.
- Hỗ trợ cả client-side và server-side rendering trong cùng một dự án.

2. So sánh các phương pháp Rendering

Client-Side Rendering (CSR)

- **Cách hoạt động**: Browser tải một file HTML gần như trống và file JavaScript lớn. React chạy trên browser, gọi API và render nội dung.
- **Ưu điểm**:
  - Điều hướng giữa các trang nhanh sau khi tải ban đầu.
  - Phù hợp với ứng dụng tương tác cao (như dashboard).
- **Nhược điểm**:
  - Thời gian tải lần đầu (First Contentful Paint - FCP) chậm.
  - SEO kém vì bot tìm kiếm chỉ thấy HTML trống.
- **Ví dụ**: Ứng dụng tạo bằng `create-react-app` hoặc Vite.

Server-Side Rendering (SSR)

- **Cách hoạt động**: Mỗi request từ client gửi đến server, server chạy React, fetch dữ liệu, render thành HTML và gửi về browser. Sau đó, JavaScript được tải để "hydrate" (kích hoạt tính tương tác).
- **Ưu điểm**:
  - SEO tốt vì bot thấy nội dung HTML hoàn chỉnh.
  - FCP nhanh hơn CSR.
  - Phù hợp với trang có nội dung động (như trang tin tức).
- **Nhược điểm**:
  - Server phải xử lý mỗi request, có thể tăng tải.
  - Điều hướng giữa các trang có thể chậm hơn CSR.
- **Hàm hỗ trợ trong Next.js**: `getServerSideProps`.

Static Site Generation (SSG)

- **Cách hoạt động**: Các trang được render thành HTML tĩnh tại thời điểm build. Server chỉ gửi file HTML đã tạo sẵn khi có request.
- **Ưu điểm**:

    - Tốc độ tải cực nhanh.
    - SEO tốt và an toàn (không cần server động).
    - Lý tưởng cho trang tĩnh như blog, tài liệu, hoặc marketing.
- **Nhược điểm**:
    - Cần build lại khi nội dung thay đổi (trừ khi dùng Incremental Static Regeneration - ISR).
    - Không phù hợp với dữ liệu thay đổi thường xuyên.
- **Hàm hỗ trợ trong Next.js**: `getStaticProps` và `getStaticPaths`.

Next.js linh hoạt: Cho phép chọn phương pháp rendering (CSR, SSR, SSG) cho từng trang trong cùng một ứng dụng.

3. File-system Based Routing

Next.js sử dụng cấu trúc thư mục để định nghĩa route, loại bỏ nhu cầu dùng thư viện như `react-router-dom`. Cụ thể:

- Thư mục `pages` (hoặc `app` trong App Router) ánh xạ trực tiếp đến URL.
- **Quy tắc**:
    - `pages/index.js` → `/`
    - `pages/about.js` → `/about`
    - `pages/blog/first-post.js` → `/blog/first-post`
- **Dynamic Routes**:
    - `pages/posts/[id].js` → `/posts/1`, `/posts/abc`,...
    - `pages/[category]/[id].js` → `/tech/123`, `/news/456`,...
- **API Routes**:
    - Thư mục `pages/api` chứa các endpoint API.
    - Ví dụ: `pages/api/hello.js` → `/api/hello`.

4. Cài đặt và cấu trúc dự án

Cài đặt Next.js

Chạy lệnh:

```
npx create-next-app@latest my-next-app
cd my-next-app
npm run dev
```

- Truy cập `http://localhost:3000` để xem ứng dụng.

Cấu trúc thư mục

- `pages/` : Chứa các file định nghĩa route.
- `public/` : Chứa tài nguyên tĩnh như hình ảnh, font.
- `styles/` : Chứa file CSS (Next.js hỗ trợ CSS modules mặc định).
- `next.config.js` : File cấu hình Next.js (ví dụ: thêm biến môi trường, cấu hình Webpack).

Pages Router vs App Router

- **Pages Router**: Dùng thư mục `pages` , dễ học, phù hợp cho người mới. Khóa học này tập trung vào Pages Router.
- **App Router**: Dùng thư mục `app` , hỗ trợ các tính năng mới như React Server Components, streaming. Sẽ được giới thiệu sơ qua để học viên định hướng.

C. Bài tập thực hành

Bài 1: Tạo dự án Next.js mới

- **Yêu cầu**: Sử dụng `create-next-app` để tạo dự án Next.js.
- **Hướng dẫn**:
    1. Chạy lệnh:

```
npx create-next-app@latest my-next-app
```

2. Chọn các tùy chọn mặc định (TypeScript: No, ESLint: Yes, Tailwind: No, App Router: No).
3. Vào thư mục và chạy:

```
cd my-next-app
npm run dev
```

4. Mở trình duyệt tại `http://localhost:3000` để kiểm tra.

Bài 2: Tạo các trang tĩnh

- **Yêu cầu**: Tạo hai trang tĩnh `/about` và `/contact`.
- **Hướng dẫn**:
  1. Tạo `pages/about.js`:

```
function AboutPage() {
  return (
    <div>
      <h1>Giới thiệu</h1>
      <p>Tôi là một lập trình viên với 10 năm kinh nghiệm trong Java
    </div>
  );
}
export default AboutPage;
```

  2. Tạo `pages/contact.js`:

```
function ContactPage() {
  return (
```

```
    <div>
      <h1>Liên hệ</h1>
      <p>Email: example@email.com</p>
      <p>Điện thoại: 0123-456-789</p>
    </div>
  );
}
export default ContactPage;
```

3. Kiểm tra tại `http://localhost:3000/about` và `http://localhost:3000/contact`.

Bài 3: Tạo component Layout

- **Yêu cầu**: Tạo component `Layout` với Header (có link điều hướng) và Footer, sử dụng trong các trang `about` và `contact`.
- **Hướng dẫn**:
  1. Tạo `components/Layout.js`:

```
import Link from 'next/link';

function Layout({ children }) {
  return (
    <div>
      <header>
        <nav>
          <Link href="/">Trang chủ</Link> | <Link href="/about">Giới
        </nav>
      </header>
      <main>{children}</main>
      <footer>
        <p>&copy; 2025 My Next App</p>
      </footer>
    </div>
  );
}
export default Layout;
```

2. Cập nhật `pages/about.js`:

```
import Layout from '../components/Layout';

function AboutPage() {
  return (
    <Layout>
      <h1>Giới thiệu</h1>
      <p>Tôi là một lập trình viên với 10 năm kinh nghiệm trong Java
    </Layout>
  );
}
export default AboutPage;
```

3. Cập nhật `pages/contact.js`:

```
import Layout from '../components/Layout';

function ContactPage() {
  return (
    <Layout>
      <h1>Liên hệ</h1>
      <p>Email: example@email.com</p>
      <p>Điện thoại: 0123-456-789</p>
    </Layout>
  );
}
export default ContactPage;
```

4. Thêm style trong `styles/globals.css`:

```
header, footer {
  background: #f4f4f4;
  padding: 10px;
  text-align: center;
```

```
}
nav a {
  margin: 0 10px;
  text-decoration: none;
  color: blue;
}
main {
  padding: 20px;
}
```

Bài 4: Chuyển đổi ứng dụng Blog sang Next.js

- **Yêu cầu**: Tạo trang danh sách bài viết (`/blog`) và trang chi tiết bài viết (`/blog/[slug]`).
- **Hướng dẫn**:
  1. Tạo `pages/blog/index.js` :

```
import Link from 'next/link';
import Layout from '../../components/Layout';

const posts = [
  { slug: 'post-1', title: 'Bài viết 1', content: 'Nội dung bài viết
  { slug: 'post-2', title: 'Bài viết 2', content: 'Nội dung bài viết
];

function BlogPage() {
  return (
    <Layout>
      <h1>Danh sách bài blog</h1>
      <ul>
        {posts.map(post => (
          <li key={post.slug}>
            <Link href={`/blog/${post.slug}`}>{post.title}</Link>
          </li>
        ))}
      </ul>
    </Layout>
  );
```

```
}
export default BlogPage;
```

2. Tạo `pages/blog/[slug].js`:

```
import Layout from '../../components/Layout';

const posts = [
  { slug: 'post-1', title: 'Bài viết 1', content: 'Nội dung bài viết
  { slug: 'post-2', title: 'Bài viết 2', content: 'Nội dung bài viết
];

function PostPage({ post }) {
  if (!post) return <p>Bài viết không tồn tại</p>;

  return (
    <Layout>
      <h1>{post.title}</h1>
      <p>{post.content}</p>
    </Layout>
  );
}

export async function getServerSideProps({ params }) {
  const post = posts.find(p => p.slug === params.slug);
  return { props: { post: post || null } };
}

export default PostPage;
```

3. Kiểm tra tại `http://localhost:3000/blog` và `http://localhost:3000/blog/post-1`.

Bài 5: Thử nghiệm với API Routes

- **Yêu cầu**: Tạo API endpoint `/api/hello` trả về JSON `{ message: 'Hello from API' }`.
- **Hướng dẫn**:

1. Tạo `pages/api/hello.js`:

```
export default function handler(req, res) {
  res.status(200).json({ message: 'Hello from API' });
}
```

2. Kiểm tra tại `http://localhost:3000/api/hello` hoặc dùng `fetch` trong trình duyệt:

```
fetch('/api/hello').then(res => res.json()).then(data => console.log
```