Chapter 07: Data Fetching trong Next.js

A. Mục tiêu

Sau bài học này, học viên sẽ:

- Nắm vững các hàm **data fetching** phía server của Next.js: `getStaticProps`, `getServerSideProps`, và `getStaticPaths`.
- Biết cách chọn phương pháp phù hợp giữa **SSG** và **SSR** dựa trên yêu cầu ứng dụng.
- Xây dựng các trang động với **SSG** sử dụng `getStaticPaths`.
- Hiểu và áp dụng **Incremental Static Regeneration (ISR)** để cập nhật nội dung tĩnh.

B. Nội dung lý thuyết

1. Tổng quan về Data Fetching trong Next.js

Next.js mở rộng React bằng cách cung cấp các phương pháp **fetch dữ liệu phía server** hoặc tại thời điểm **build**. Điều này giúp:

- Cải thiện **SEO** bằng cách cung cấp HTML hoàn chỉnh cho bot tìm kiếm.
- Giảm thời gian tải ban đầu so với **Client-Side Rendering (CSR)**.
- Tăng tính linh hoạt khi kết hợp các phương pháp rendering trong cùng một dự án.

Các hàm chính:

- `getStaticProps`: Dùng cho **Static Site Generation (SSG)**.
- `getServerSideProps`: Dùng cho **Server-Side Rendering (SSR)**.
- `getStaticPaths`: Dùng cho các trang động trong SSG.

2. getStaticProps (Static Site Generation - SSG)

- **Cách hoạt động**: Hàm chạy tại thời điểm **build** trên server, fetch dữ liệu và render trang thành HTML tĩnh.
- **Cú pháp**:

```
export async function getStaticProps(context) {
  const data = await fetchData();
  return {
    props: { data }, // Truyền vào component qua props
    revalidate: 60, // Optional: ISR sau 60 giây
  };
}
```

- **Context object**: Chứa `params` (cho dynamic routes), `preview` (cho chế độ xem trước).
- **Giá trị trả về**:
  - `{ props: { ... } }`: Dữ liệu truyền vào component.
  - `{ notFound: true }`: Hiển thị trang 404.
  - `{ redirect: { destination: '/new-url', permanent: false } }`: Chuyển hướng.
- **Khi nào sử dụng**: Dữ liệu tĩnh, không thay đổi sau mỗi request (blog, sản phẩm, tài liệu).
- **Ví dụ**:

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

function Page({ data }) {
  return <div>{data.title}</div>;
}
export default Page;
```

3. getServerSideProps (Server-Side Rendering - SSR)

- **Cách hoạt động**: Hàm chạy trên mỗi **request**, fetch dữ liệu mới nhất, render HTML và gửi về client.
- **Cú pháp**:

```
export async function getServerSideProps(context) {
  const { req, res, query, params } = context;
  const data = await fetchData();
  return { props: { data } };
}
```

- **Context object**: Bao gồm `req`, `res`, `query`, `params`,...
- **Giá trị trả về**: Tương tự `getStaticProps`.
- **Khi nào sử dụng**: Dữ liệu thay đổi liên tục, phụ thuộc vào người dùng (dashboard, giỏ hàng).
- **Ví dụ**:

```
export async function getServerSideProps({ params }) {
  const res = await fetch(`https://api.example.com/users/${params.id}`);
  const user = await res.json();
  return { props: { user } };
}

function Profile({ user }) {
  return <div>{user.name}</div>;
}
export default Profile;
```

4. getStaticPaths (Cho Dynamic SSG Pages)

- **Vấn đề**: Với trang động như `pages/posts/[id].js`, Next.js cần biết những giá trị `id` nào để render trước.
- **Cách hoạt động**: Hàm chạy tại thời điểm **build**, tạo danh sách các đường dẫn cần render.

- **Cú pháp**:

```
export async function getStaticPaths() {
  const res = await fetch('https://api.example.com/posts');
  const posts = await res.json();
  const paths = posts.map(post => ({
    params: { id: post.id.toString() },
  }));
  return { paths, fallback: false };
}
```

- **Giá trị trả về**:
  - `paths` : Mảng các object `{ params: { key: value } }` .
  - `fallback` :
    - `false` : 404 cho các path không được liệt kê.
    - `true` : Hiển thị trạng thái "fallback" và render động.
    - `'blocking'` : Chờ server render xong trước khi hiển thị.
- **Khi nào sử dụng**: Kết hợp với `getStaticProps` cho trang động trong SSG.
- **Ví dụ**:

```
export async function getStaticPaths() {
  return {
    paths: [{ params: { id: '1' } }, { params: { id: '2' } }],
    fallback: false,
  };
}

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

5. Incremental Static Regeneration (ISR)

- **Cách hoạt động**: Kết hợp tốc độ của SSG với dữ liệu tươi mới. Trang được render tĩnh tại thời điểm build, nhưng Next.js tự động cập nhật sau một khoảng thời gian (`revalidate`).
- **Cú pháp**:

```
export async function getStaticProps() {
  const res = await fetch('https://api.example.com/data');
  const data = await res.json();
  return {
    props: { data },
    revalidate: 10, // Cập nhật sau 10 giây
  };
}
```

- **Lợi ích**:
  - Phục vụ trang tĩnh nhanh chóng.
  - Tự động cập nhật dữ liệu mà không cần build lại toàn bộ site.
- **Khi nào sử dụng**: Blog, danh sách sản phẩm, trang tin tức không cần dữ liệu thời gian thực.

C. Bài tập thực hành

Bài 1: SSG - Danh sách bài viết

- **Yêu cầu**: Tạo trang `/posts` hiển thị danh sách tiêu đề bài viết từ API `https://jsonplaceholder.typicode.com/posts` bằng `getStaticProps` .
- **Hướng dẫn**:
  1. Tạo `pages/posts/index.js` :

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

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

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

export default PostsPage;
```

2. Kiểm tra tại `http://localhost:3000/posts`.

Bài 2: Dynamic SSG - Chi tiết bài viết

- **Yêu cầu**: Tạo trang chi tiết bài viết `/posts/[id]` với `getStaticPaths` và `getStaticProps`, đặt `fallback: false`.
- **Hướng dẫn**:
  1. Tạo `pages/posts/[id].js`:

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

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

  return (
    <Layout>
```

```
      <h1>{post.title}</h1>
      <p>{post.body}</p>
    </Layout>
  );
}

export async function getStaticPaths() {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts
  const posts = await res.json();
  const paths = posts.map(post => ({
    params: { id: post.id.toString() },
  }));
  return { paths, fallback: false };
}

export async function getStaticProps({ params }) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts
  const post = await res.json();
  return { props: { post } };
}

export default PostPage;
```

2. Kiểm tra tại `http://localhost:3000/posts/1`.

Bài 3: SSR - Trang hồ sơ người dùng

- **Yêu cầu**: Tạo trang `/profile` sử dụng `getServerSideProps` để fetch dữ liệu từ `https://randomuser.me/api/`.
- **Hướng dẫn**:
  1. Tạo `pages/profile.js`:

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

function ProfilePage({ user }) {
  return (
    <Layout>
      <h1>Hồ sơ người dùng</h1>
      <img src={user.picture.large} alt="Avatar" />
```

```
      <p>Họ tên: {user.name.first} {user.name.last}</p>
      <p>Email: {user.email}</p>
    </Layout>
  );
}

export async function getServerSideProps() {
  const res = await fetch('https://randomuser.me/api/');
  const data = await res.json();
  const user = data.results[0];
  return { props: { user } };
}

export default ProfilePage;
```

2. Kiểm tra tại `http://localhost:3000/profile`. Tải lại trang để thấy dữ liệu mới.

Bài 4: Fallback với getStaticPaths

- **Yêu cầu**: Chỉ render trước 10 bài viết đầu tiên trong `getStaticPaths`, đặt `fallback: true`, hiển thị trạng thái "Loading..." khi truy cập bài viết ngoài danh sách.
- **Hướng dẫn**:
  1. Cập nhật `pages/posts/[id].js`:

```
import { useRouter } from 'next/router';
import Layout from '../../components/Layout';

function PostPage({ post }) {
  const router = useRouter();

  if (router.isFallback) {
    return <div>Loading...</div>;
  }

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

  return (
    <Layout>
      <h1>{post.title}</h1>
```

```
      <p>{post.body}</p>
    </Layout>
  );
}

export async function getStaticPaths() {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts
  const posts = await res.json();
  const paths = posts.slice(0, 10).map(post => ({
    params: { id: post.id.toString() },
  }));
  return { paths, fallback: true };
}

export async function getStaticProps({ params }) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts
  const post = await res.json();
  return { props: { post } };
}

export default PostPage;
```

2. Kiểm tra tại `http://localhost:3000/posts/11`. Trang sẽ hiển thị "Loading..." trước khi render nội dung.

Bài 5: ISR - Cập nhật danh sách bài viết

- **Yêu cầu**: Thêm `revalidate: 10` vào `getStaticProps` của trang `/posts`. Kiểm tra cập nhật dữ liệu sau 10 giây.
- **Hướng dẫn**:
  1. Cập nhật `pages/posts/index.js`:

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

function PostsPage({ posts }) {
  return (
    <Layout>
```

```
      <h1>Danh sách bài viết</h1>
      <ul>
        {posts.map(post => (
          <li key={post.id}>
            <Link href={`/posts/${post.id}`}>{post.title}</Link>
          </li>
        ))}
      </ul>
    </Layout>
  );
}

export async function getStaticProps() {
  const res = await fetch('https://jsonplaceholder.typicode.com/post
  const posts = await res.json();
  return {
    props: { posts },
    revalidate: 10, // Cập nhật sau 10 giây
  };
}

export default PostsPage;
```

2. Build và start dự án:

```
npm run build
npm run start
```

3. Truy cập `http://localhost:3000/posts`. Dùng Postman để gửi POST request thêm bài viết mới vào API (nếu API hỗ trợ, hoặc mô phỏng bằng cách chỉnh sửa dữ liệu cục bộ). Tải lại trang sau 10 giây để thấy thay đổi.