Theory 12
Data Fetching, Styling, SEO, Error Handling

WEEK 04

Nội dung chính

CSS Styling

❑ **CSS trong Next.js:**

❍ **Next.js** có hỗ trợ tích hợp cho kiểu **JSX**, nhưng bạn cũng có thể sử dụng các thư viện **CSS-in-JS** phổ biến như **Styled Component** hoặc **emotion**.

❍ **Style-jsx** là thư viện **CSS-in-JS** cho phép bạn viết CSS trong React component và các kiểu CSS sẽ được xác định phạm vi (các thành phần khác sẽ không bị ảnh hưởng).

❍ **Next.js** có hỗ trợ tích hợp cho **CSS** và **SASS**, cho phép bạn nhập các tệp **.css** và **.scss**

CSS Styling

- **Tạo Layout Component:**
  - **Layout component** có nhiệm vụ tạo layout chung cho các trang.
  - Code tạo component layout:

```
export default function Layout({ children }) {
    return <div>{children}</div>
}
```

CSS Styling

❑ **Sử dụng Layout Component:**

```
export default function Layout({ children }) {
    return (
        <Layout>
            <Head><title>First Post</title></Head>
            <h1>First Post</h1>
            <h2><Link href="/"><a>Back to home</a></Link></h2>
        </Layout>
    );
}
```

CSS Styling

- **Thêm style cho Layout Component:**
  - Mô đun CSS cho phép bạn import file CSS trong một React Component.
  - Tạo một file CSS có tên là layout.module.css.
  - Để sử dụng mô đun CSS, tên file phải được kết thúc với đuôi .module.css.
  - Sử dụng style:

```
import styles from “./layout.module.css”;
export default function Layout({ children }) {
  return (
      <div className={styles.container}>{children}</div>
   );
}
```

CSS Styling

- **Global style:**
  - Khi bạn muốn CSS được load cho mọi trang, Next.js hỗ trợ bạn với CSS global.
  - Bạn có thể đặt file global CSS ở bất kỳ đâu và sử dụng bất kỳ tên nào.
  - Thêm các tệp global CSS bằng cách import chúng trong **pages/_app.js**
  - Sử dụng style:

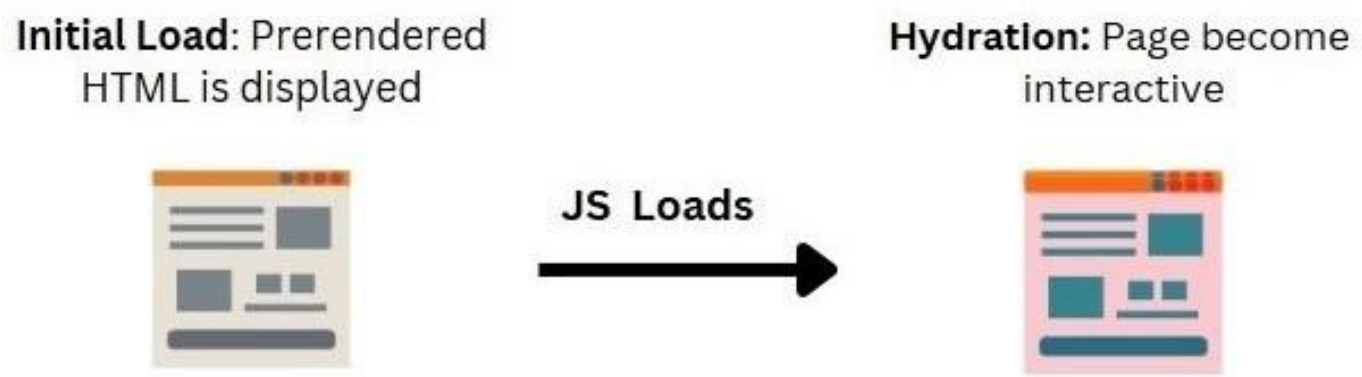
```
import “../styles/global.css”;
export default function App({ Component, pageProps }) {
  return (
      <Component {...pageProps} />
   );
}
```

Data Fetching

❑ **Khái niệm Pre-rendering:**

❍ **Next.js** sẽ render lại ở mỗi trang, nghĩa là HTML sẽ được khởi tạo ở mỗi page thay vì sử dụng javascript để xử lý phía client.

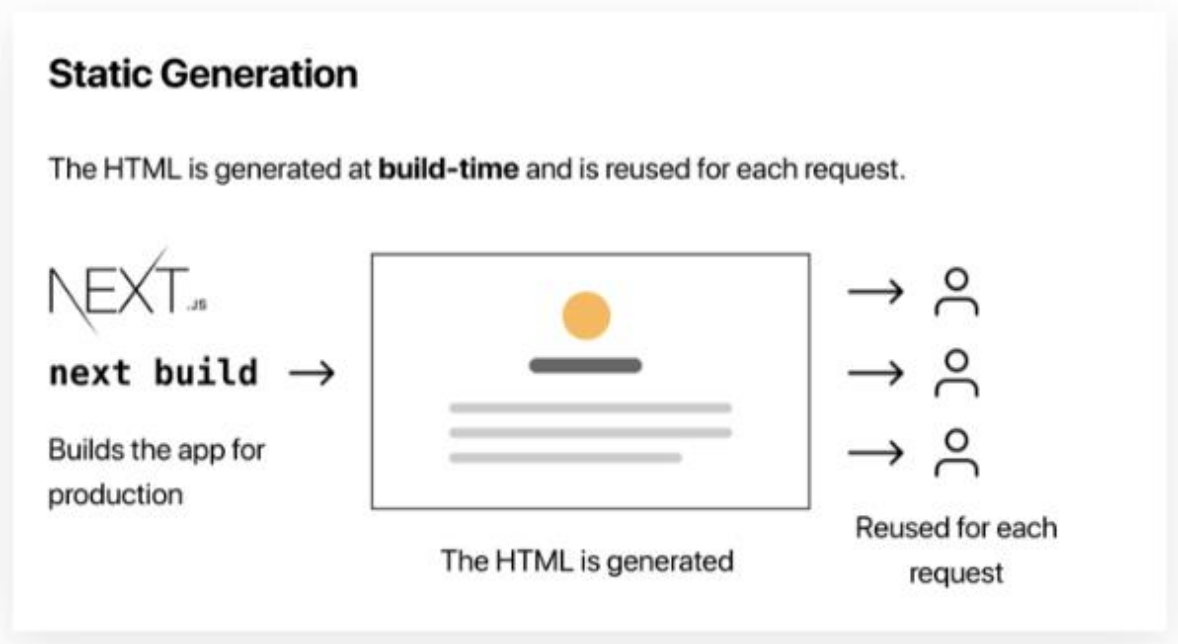
❍ **Pre-rendering** sẽ tốt hơn cho **performance** và **SEO**.



Data Fetching

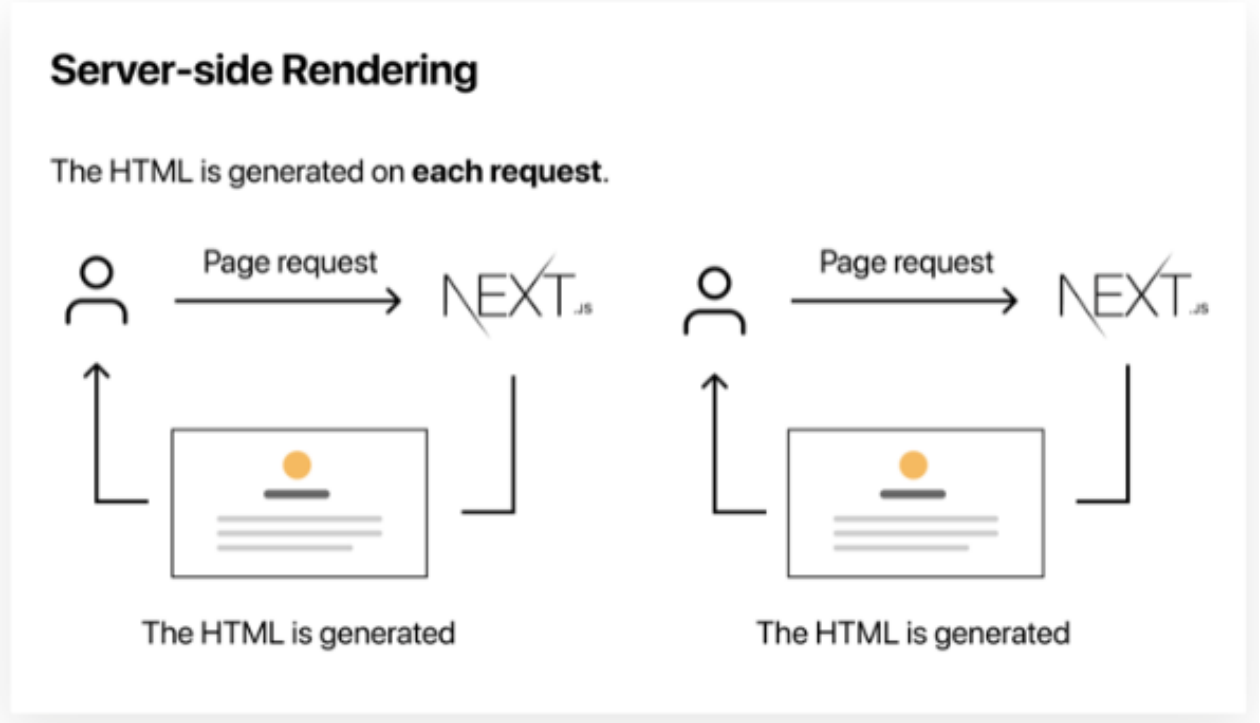
❑ **Hai hình thức Pre-rendering:**

❍ **Static Generation:** HTML sẽ được khởi tạo ngay tại thời điểm build app (npm run build). HTML được render trước và sử dụng lại theo từng request.



Data Fetching

- **Hai hình thức Pre-rendering:**
  - **Server-Side Rendering:** khởi tạo HTML trên mỗi request.



Data Fetching

❑ **Nên sử dụng hình thức render nào?**

❍ Sử dụng **Static Generation** (có và không có dữ liệu) bất cứ khi nào có thể vì trang của bạn được tạo một lần và được CDN phân phát, điều này làm cho dữ liệu được tải nhanh hơn nhiều so với việc máy chủ hiển thị trang theo yêu cầu.

❍ Nếu trang của bạn cần cập nhật dữ liệu thường xuyên, nội dung trang thay đổi trên mỗi request, hãy sử dụng **Server-Side Rendering**.

❍ **Next.js** cho phép bạn tùy chọn kiểu **pre-rendering** cho mỗi trang.

Data Fetching

❑ getStaticProps (Static Generation)

❍ Phương thức **getStaticProps** có thể được sử dụng bên trong một Page để lấy dữ liệu ngay tại thời điểm build. Mỗi khi ứng dụng đã được build, nó sẽ không làm mới dữ liệu cho đến khi một bản build khác được khởi động.

```
export async function getStaticProps(context) {
    //fetch data từ file system, API, DB,...
    const data = ...

    // Giá trị của props sẽ được truyền tới component Home
    return {  props: ... }
}

export default function Home(props) { ... }
```

Data Fetching

❑ **getServerSideProps (Server-Side Rendering)**

❍ Phương thức **getServerSideProps** lấy dữ liệu mỗi khi user gửi request lên hệ thống. Nó sẽ tìm, nạp dữ liệu trước khi client có thể view được trang Web. Nếu client đưa ra các request tiếp theo, dữ liệu sẽ được tìm và nạp lại.

```
export async function getServerSideProps(context) {
    return {
        props: {
            // Giá trị của props cho component của bạn
        }
    }
}
```

SEO Optimization

- **Các yếu tố ảnh hưởng đến SEO của Web App:**
  - Title, Meta, Open Graph, cấu trúc URL, tốc độ load,...
  - Sử dụng sitemap.xml, robots.txt, canonical URL.

SEO Optimization

- **Ưu điểm của Next.js hỗ trợ SEO tốt:**
  - Server-side Rendering (SSR)
  - Static Site Generation (SSG)
  - Dynamic routing thân thiện SEO
  - Tích hợp tốt với Open Graph, JSON-LD (linked data)

SEO Optimization

- **Cách dùng next/head để tối ưu SEO:**
  - Sử dụng component Head thay thẻ head của HTML
  - Dùng cho mỗi trang để tùy chỉnh metadata riêng biệt

```
import Head from 'next/head';

<Head>
    <title>My Blog - Học Next.js</title>
    <meta name="description" content="Khóa học Next.js từ cơ bản đến nâng cao." />
    <meta property="og:title" content="My Blog" />
    <meta property="og:image" content="/cover.png" />
</Head>
```

SEO Optimization

- **Sử dụng metadata động theo bài viết để tối ưu SEO:**
  - Lấy metadata theo nội dung động (SSG hoặc SSR)

```
<Head>
    <title>{post.title}</title>
    <meta name="description" content={post.excerpt} />
</Head>
```

SEO Optimization

- **Sử dụng next-sitemap để tạo sitemap.xml, robots.txt để tối ưu SEO:**
  - Cài đặt next-sitemap: npm install next-sitemap
  - Tạo file next-sitemap.config.js tại root có nội dung như sau:

```
/** @type {import('next-sitemap').IConfig} */
module.exports = {
   siteUrl: 'https://your-domain.com', // replace your domain
   generateRobotsTxt: true,  // create robots.txt
   changefreq: 'weekly',
   priority: 0.7,
   sitemapSize: 5000,
   exclude: ['/admin/*'], // if need
};
```

SEO Optimization

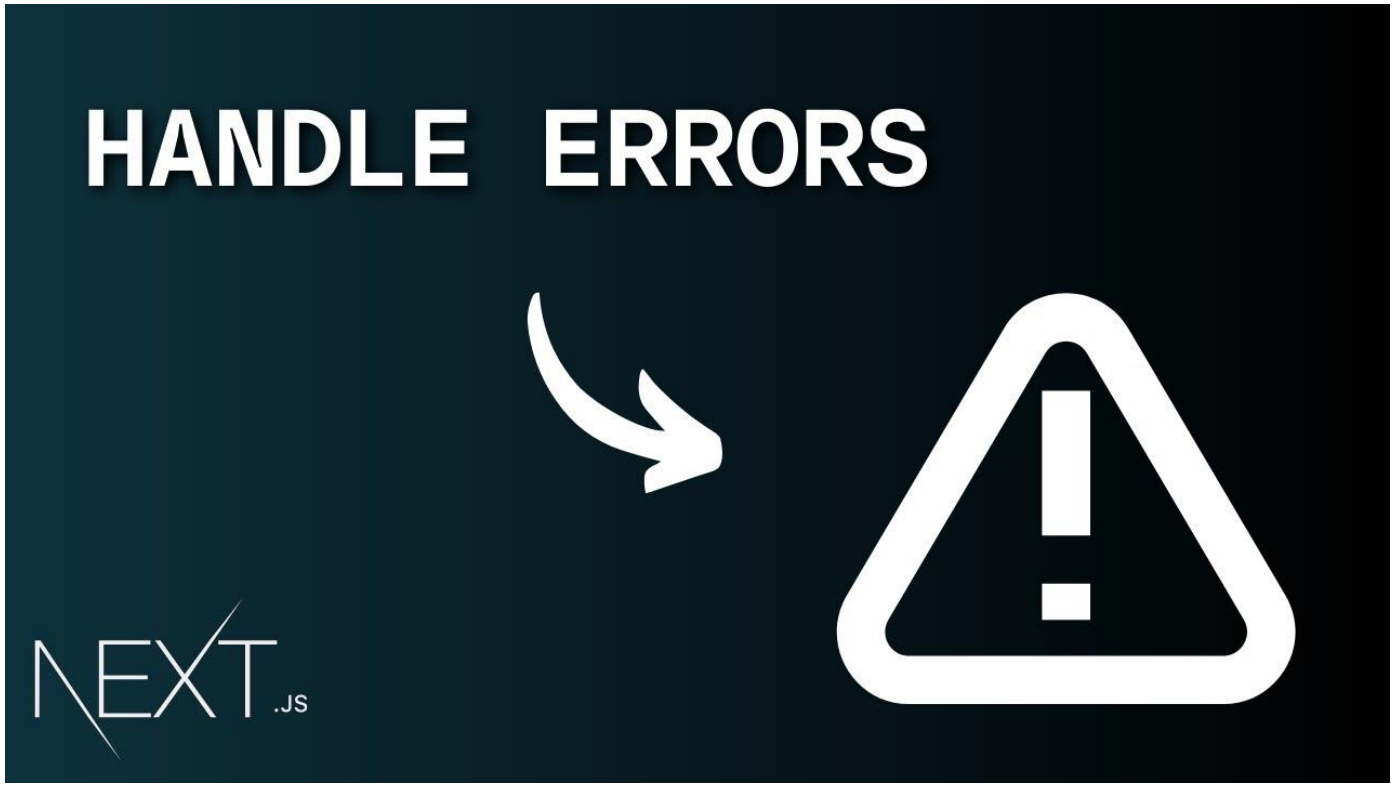
- **Sử dụng next-sitemap để tạo sitemap.xml, robots.txt để tối ưu SEO:**
  - Cập nhật **package.json** để build sitemap sau khi build project:

```
{
    "scripts": {
        "build": "next build",
        "postbuild": "next-sitemap"
    }
}
```

  - Sai khi chạy npm run build, sitemap.xml và robots.txt sẽ được tạo trong thư mục public.
  - Đăng sitemap lên Google Search Console:
    https://search.google.com/search-console

Error Handling

- **Các loại lỗi phổ biến:**
  - 404 Not Found
  - 500 Internal Server Error
  - Lỗi khi gọi API
  - Lỗi khi render phía client



Error Handling

- **Custom trang 404:**
  - Tạo file:
    /app/not-found.tsx (App Router)
    /pages/404.jsx (Pages Router)

  - Ví dụ:

```
export default function NotFound() {
    return <h1>Trang bạn tìm không tồn tại</h1>;
}
```

Error Handling

- **Custom trang 500 (App Router):**
  - Tạo file:
    /app/error.tsx (App Router)

  - Ví dụ:

```
'use client';

export default function Error({ error, reset }) {
        return (
                <div><h2>Đã có lỗi xảy ra!</h2><p>{error.message}</p></div>;
        );
}
```

Error Handling

❑ **try-catch khi fetch data từ API:**

```
try {
        const res = await fetch('https://api.example.com/posts');
        if (!res.ok) throw new Error('Lỗi khi fetch dữ liệu');
        const data = await res.json();
} catch (err) {
        console.error(err.message);
}
```

Error Handling

- **Error.tsx trong từng layout:** Bạn cũng có thể tạo **error.tsx** trong từng folder **app/xxx/** để xử lý lỗi riêng cho từng phần.
- Một số lưu ý khi xử lý lỗi:
  - Luôn kiểm tra **res.ok** khi gọi API
  - Dùng **fallback** cho component bất đồng bộ
  - Log lỗi server riêng biệt với **middleware** hoặc logging service
  - Tránh lộ **stack trace** ra UI production

Tóm tắt bài học

- ❑ CSS Styling
- ❑ Data Fetching
- ❑ SEO Optimization
- ❑ Error Handling