Next js
Lecture 2

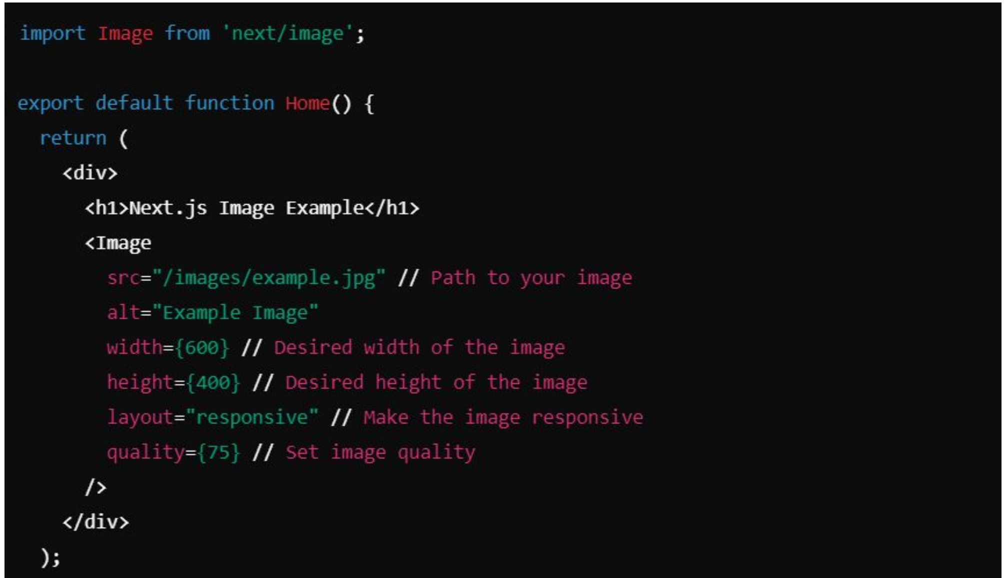
By Mona Soliman

Agenda

- metadata & Generating Dynamic metadata
- Fonts
- Image
- Nested Routes and pages & Nested Layouts
- Static vs Dynamic rendering
- Static Site Generation (ssg)
- Incremental Static Regeneration (ISR)
- working with mongoose (get , post)

IMAGE

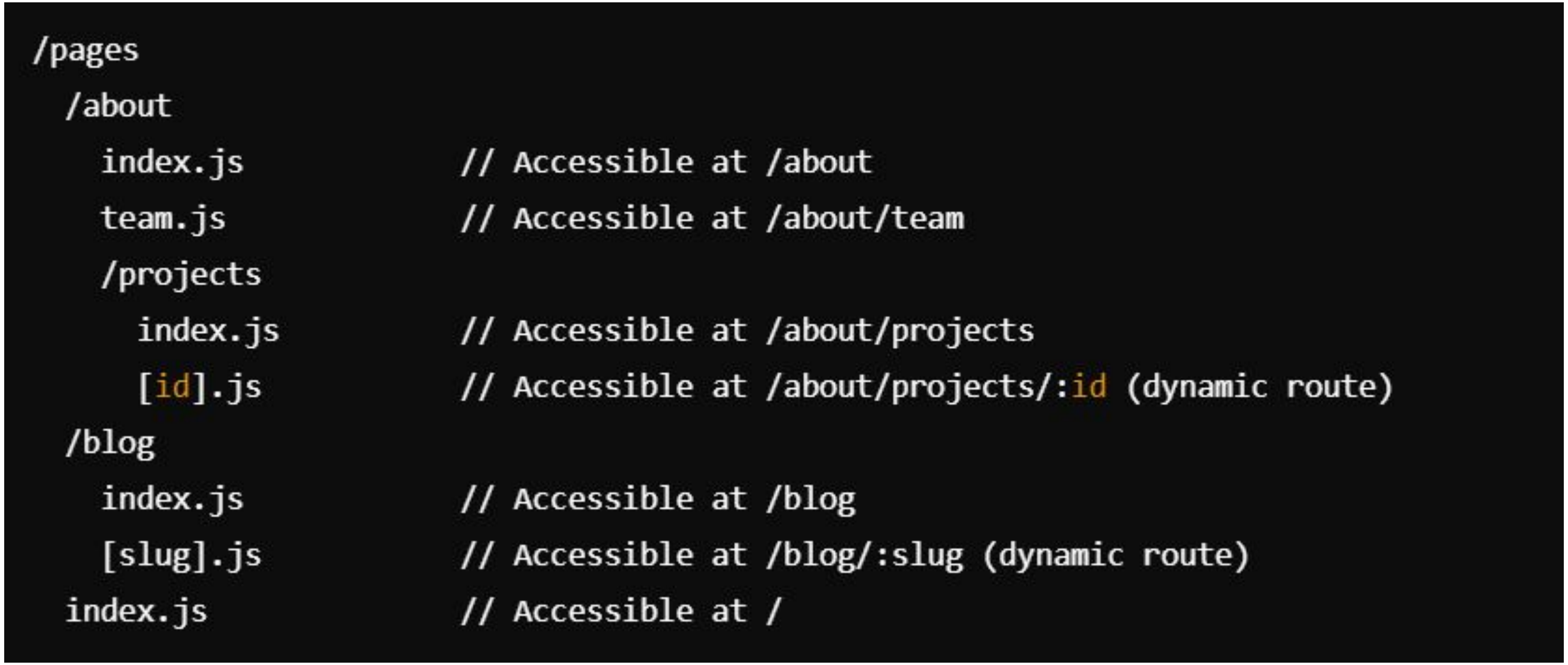
The Image component in Next.js is a built-in component provided by Next.js to optimize and manage images in your web applications automatically. It is part of the Next.js next/image package and offers several benefits over the traditional <img> HTML tag, such as improved performance, automatic optimization, responsive resizing, lazy loading, and more.



```
import Image from 'next/image';

export default function Home() {
  return (
    <div>
      <h1>Next.js Image Example</h1>
      <Image
        src="/images/example.jpg" // Path to your image
        alt="Example Image"
        width={600} // Desired width of the image
        height={400} // Desired height of the image
        layout="responsive" // Make the image responsive
        quality={75} // Set image quality
      />
    </div>
  );
```

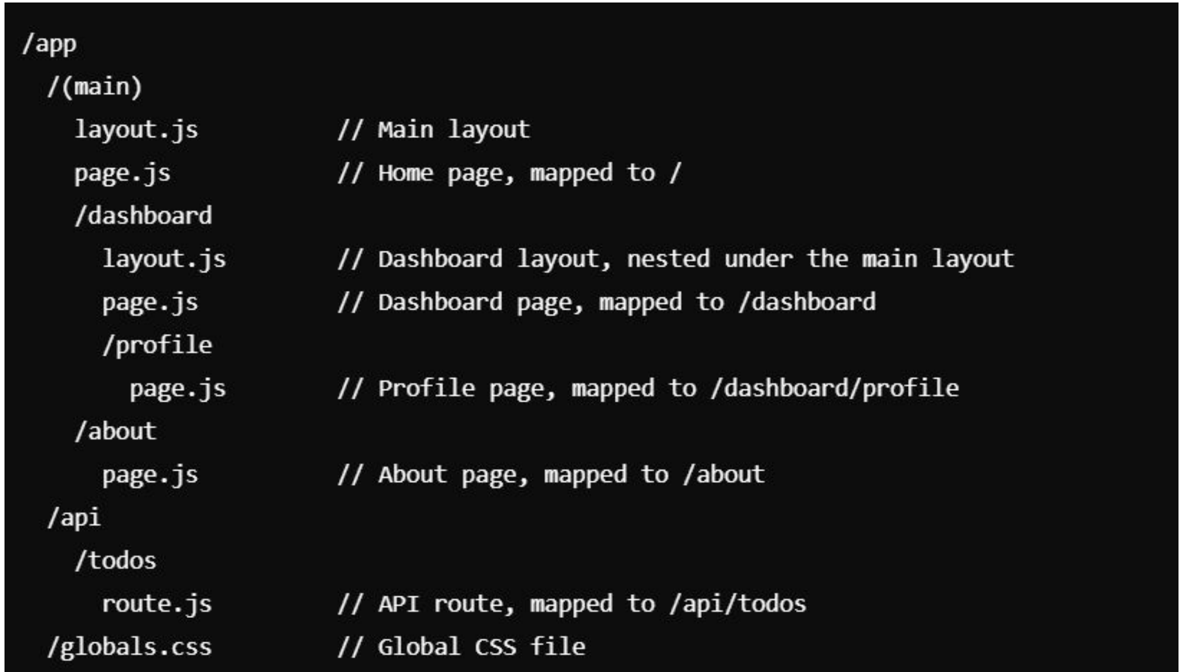
Nested Routes



```
/pages
  /about
    index.js            // Accessible at /about
    team.js             // Accessible at /about/team
    /projects
      index.js          // Accessible at /about/projects
      [id].js           // Accessible at /about/projects/:id (dynamic route)
  /blog
    index.js            // Accessible at /blog
    [slug].js           // Accessible at /blog/:slug (dynamic route)
  index.js              // Accessible at /
```

Nested Layouts

Layouts are defined at different levels in the app directory, allowing you to create persistent layouts for nested pages, like sidebars or headers that remain consistent across nested routes.



```
/app
  /(main)
    layout.js          // Main layout
    page.js            // Home page, mapped to /
    /dashboard
      layout.js        // Dashboard layout, nested under the main layout
      page.js          // Dashboard page, mapped to /dashboard
      /profile
        page.js        // Profile page, mapped to /dashboard/profile
    /about
      page.js          // About page, mapped to /about
  /api
    /todos
      route.js         // API route, mapped to /api/todos
  /globals.css         // Global CSS file
```

Linking & Navigating

-Using the <Link> Component

<Link> is a built-in component that extends the HTML <a> tag to provide prefetching and client-side navigation between routes. It is the primary and recommended way to navigate between routes in Next.js.

-Using the useRouter hook (Client Components)

The useRouter hook allows you to programmatically change routes from Client Components.

-Using the redirect function (Server Components)

For Server Components, use the redirect function instead.

Static vs Dynamic Rendering

STATIC RENDERING

HTML is generated at build time, or periodically in the background by re-fetching data

👉 Useful when data doesn't change often and is not personalized to user (e.g. product page)

👉 Default rendering strategy in Next.js (even when a page or component fetches data)

👉 When deployed to Vercel, each static route is automatically hosted on a CDN

👉 If all routes of an app are static, the entire app can be exported as a static site (SSG)

Static vs Dynamic Rendering

DYNAMIC RENDERING:

HTML is generated at request time (for each new request reaches the server)

👉 Makes sense if:

1- The data changes frequently and is personalized to the user (e.g. cart)

2- Rendering a route requires information that depends on request (e.g. search params)

Static vs Dynamic Rendering

👉 A route automatically switches to dynamic rendering in certain conditions :

1- The route has a dynamic segment (page uses params)

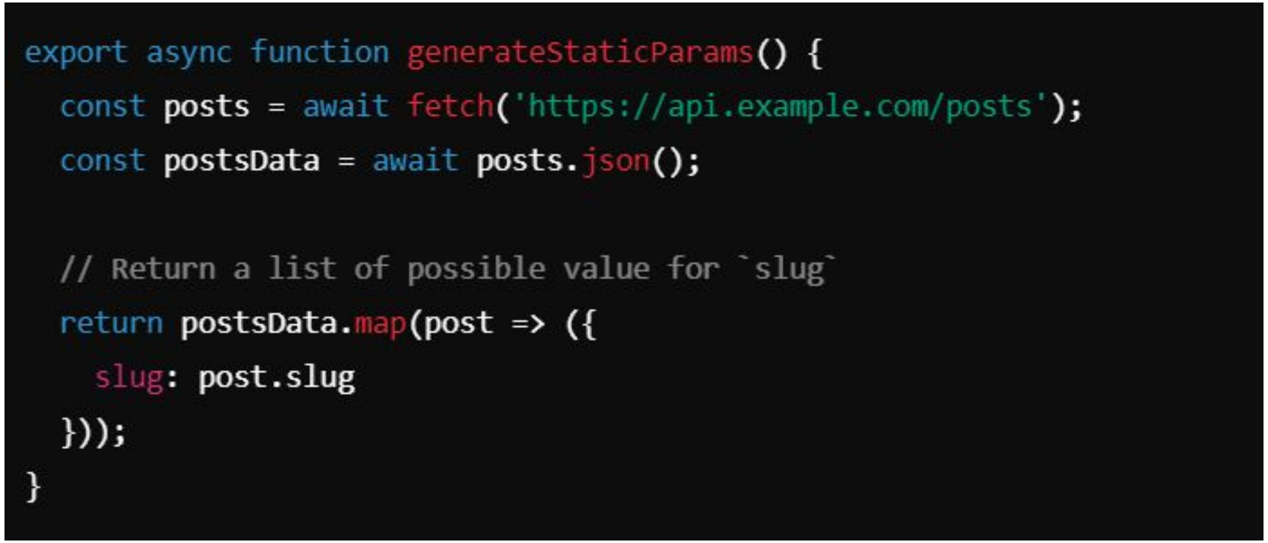
2- searchParams are used in the page component /product?quantity=23

3- headers() or cookies() are used in any of the route’s server components

4- An uncached data request is made in any of the route’s server components

Making Dynamic routes static

In Next.js 13 and later, generateStaticParams is used to create static pages at build time. This allows you to pre-render pages based on dynamic routes with the data available at build time, which can enhance performance and SEO.



```
export async function generateStaticParams() {
  const posts = await fetch('https://api.example.com/posts');
  const postsData = await posts.json();

  // Return a list of possible value for `slug`
  return postsData.map(post => ({
    slug: post.slug
  }));
}
```

Static Site Generation

Static Site Generation (SSG) in Next.js is a method of pre-rendering pages at build time. This approach allows you to generate HTML for your pages during the build process rather than at request time, which can greatly improve performance and SEO.

How Static Site Generation Works :

- **Build Time Rendering:** During the build process, Next.js generates static HTML for each page based on the data available at that time.
- **Static HTML Output:** The generated HTML files are served directly to users from a CDN, which is very fast compared to generating the HTML on-demand.
- **Data Fetching:** Data required to render the pages is fetched at build time using functions.

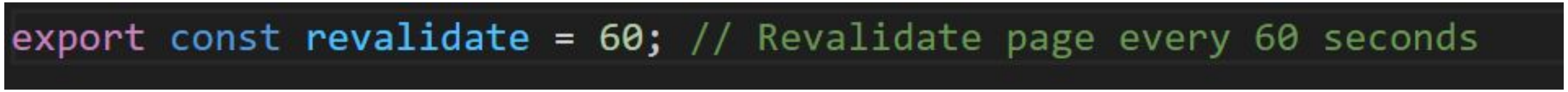
Incremental Site Regeneration

A Next.js feature that allows developers to update the content of a static page, in the background, even after the website has already been built and deployed. This happens by re-fetching the data of a component or entire route after a certain interval.

Up-to-Date Content: Ensures that your static pages are regularly updated with fresh data without a complete rebuild.

Improved Performance: Serves static pages quickly to users while regenerating content in the background.

Scalability: Efficiently handles large volumes of traffic without rebuilding the entire site.



```
export const revalidate = 60; // Revalidate page every 60 seconds
```

Working with mongodb

Let's code

Lab

LAB

1- Use mongoose to handle GET & POST users

2- change your api from jsonplaceholder to your local api

3- try to make different layouts (login without header and footer)

5- convert your dynamic routes into static

6- try to implement incremental site regeneration in users route

Bonus:
-implement localization in your app (en , ar)
-Each user has image (implement upload image)   cloudinary

Thank you