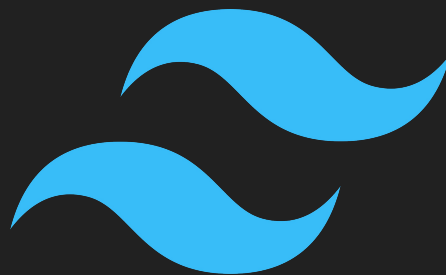
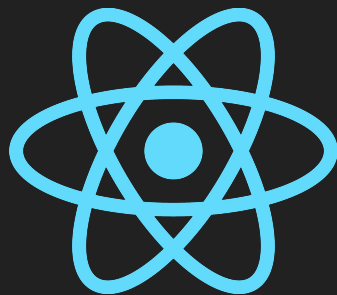


# **From Concepts to Code: Mastering NextJS and Tailwind at GIAIC**

Lead Teachers: Hamza Ahmad Alvi, Fahad Khan



**Presentation By: Abdul Karim (260099)**

# Understanding NextJS

- Next.js is a React framework for building full-stack web applications. You use React Components to build user interfaces, and Next.js for additional features and optimizations.
- Under the hood, Next.js also abstracts and automatically configures tooling needed for React, like bundling, compiling, and more. This allows you to focus on building your application instead of spending time with configuration.
- Whether you're an individual developer or part of a larger team, Next.js can help you build interactive, dynamic, and fast React applications.

# Understanding Components in NextJS

## Definition:

- Components are **reusable, self-contained pieces of UI** in React and Next.js.
- They can be as simple as a button or as complex as an entire page section, encapsulating their logic, styles, and functionality.

# Why Use Components?

## Reusability:

- Build once, use anywhere! Components allow you to reuse code across multiple pages, keeping your codebase DRY (Don't Repeat Yourself).

## Maintainability:

- With isolated components, it's easier to update, debug, and refactor code. Each component can be maintained independently, which improves scalability.

## Consistency:

- Using the same component across your app ensures a consistent look and feel, enhancing user experience.

## Improved Readability:

- Components organize your code by breaking down complex UIs into smaller, readable parts, making it easier to understand and manage.


# The `page.tsx` File

A page is UI that is unique to a route. You can define a page by default exporting a component from a `page.tsx` file.

For example, to create your index page, add the `page.js` file inside the `app` directory:



Here's an example page component:

 app/page.tsx

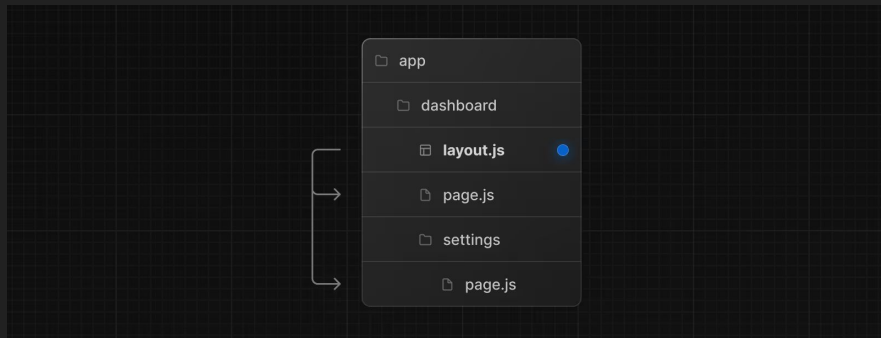
```
1  // `app/page.tsx` is the UI for the `/` URL
2  export default function Page() {
3    return <h1>Hello, Home page!</h1>
4  }
```

# The `layout.tsx` File

A layout is UI that is shared between multiple routes. On navigation, layouts preserve state, remain interactive, and do not re-render. Layouts can also be nested.

You can define a layout by default exporting a React component from a `layout.js` file. The component should accept a `children` prop that will be populated with a child layout (if it exists) or a page during rendering.

For example, the layout will be shared with the `/dashboard` and `/dashboard/settings` pages:



Here's what a basic layout .tsx might look like:

TS app/dashboard/layout.tsx

TypeScript

```
1  export default function DashboardLayout({
2    children, // will be a page or nested layout
3  }: {
4    children: React.ReactNode
5  }) {
6    return (
7      <section>
8        {/* Include shared UI here e.g. a header or sidebar */}
9        <nav></nav>
10
11        {children}
12      </section>
13    )
14  }
```



# The <Link> Component

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

Basic usage:

**TS** app/page.tsx

```
1  import Link from 'next/link'
2
3  export default function Page() {
4    return <Link href="/dashboard">Dashboard</Link>
5  }
```

# Why use `<Link>` Component

- The `Link` component enables smooth transitions between pages without full-page reloads, providing a Single Page Application (SPA) experience.
- Since Next.js prefetches linked pages, pages often load instantly, improving the overall speed and responsiveness of the app.
- Users enjoy a faster, more seamless browsing experience, with less wait time between page loads.
- The `Link` component is integrated with Next.js' routing system, simplifying navigation setup without additional libraries.
- Next.js automatically splits code for each route, loading only the code needed for the page, further boosting performance when using the `Link` component.

# How can we apply CSS in Next.js?

Next.js supports multiple ways of handling CSS, including:

- CSS Modules
- Global Styles
- External Stylesheets


# CSS Modules

Next.js has built-in support for CSS Modules using the `.module.css` extension.

CSS Modules locally scope CSS by automatically creating a unique class name. This allows you to use the same class name in different files without worrying about collisions. This behavior makes CSS Modules the ideal way to include component-level CSS.

# Example of a CSS Module

CSS Modules can be imported into any file inside the app directory:

 app/dashboard/styles.module.css

```
1  .dashboard {  
2    padding: 24px;  
3  }
```

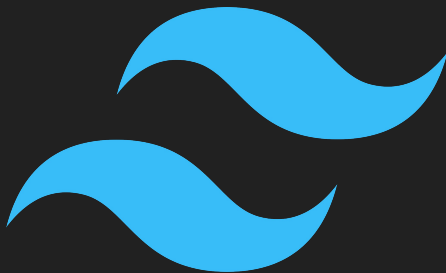
```
// app/dashboard/layout.tsx  
import styles from "../styles.module.css";  
  
export default function DashboardLayout({  
  children,  
}: {  
  children: React.ReactNode;  
}) {  
  return <section className={styles.dashboard}>{children}</section>;  
}
```

# Tailwind CSS

A utility-first CSS framework packed with classes like `flex`, `pt-4`, `text-center` and `rotate-90` that can be composed to build any design, directly in your markup.

Tailwind CSS works by scanning all of your HTML files, JavaScript components, and any other templates for class names, generating the corresponding styles and then writing them to a static CSS file.

It's fast, flexible, and reliable — with zero-runtime.



# Tailwind CSS vs. Standard CSS

## Tailwind CSS:

- **Utility-First Framework:**
  - Provides a set of predefined utility classes (e.g., `bg-blue-500`, `p-4`) for rapid styling.
- **No Custom Stylesheets Needed:**
  - Styles are applied directly to HTML elements via class names, minimizing the need for separate CSS files.
- **Responsive Design Simplified:**
  - Easily create responsive designs with classes like `md:text-lg` or `sm:bg-red-400`.
- **Pros:**
  - Faster prototyping, minimal CSS files, consistent design system.
- **Cons:**
  - HTML can become cluttered with multiple class names; requires familiarity with the framework's utility classes.

# Tailwind CSS vs. Standard CSS

## Standard CSS:

- **Traditional Styling Approach:**
  - Styles are written in custom CSS files and linked to components or pages using class names or IDs.
- **Global and Scoped Styles:**
  - Allows for both global styles (e.g., `globals.css`) and component-scoped styles (e.g., CSS Modules) in Next.js.
- **More Control Over Styles:**
  - Gives developers complete control over styles, animations, and selectors.
- **Pros:**
  - Highly customizable, widely supported, compatible with complex design needs.
- **Cons:**
  - Can lead to large, hard-to-manage CSS files; responsiveness requires additional setup (e.g., media queries).



# Wrapping Up

## Summary of Key Concepts:


- **page.tsx & layout.tsx:**
  - Central to the new App Router in Next.js 13, helping structure and layout pages with ease.
- **Link Component:**
  - Enables smooth client-side navigation, enhancing performance and user experience.
- **Reusable Components:**
  - Encourages code reusability, consistency, and cleaner organization across your app.
- **Styling Approaches:**
  - Next.js supports CSS Modules, global CSS, and frameworks like Tailwind CSS to cater to diverse styling needs.
- **Tailwind CSS vs. Standard CSS:**
  - Tailwind offers quick styling with utility classes, while standard CSS provides more flexibility for custom designs.


# Thank You


Learning at GIAIC has been a great experience and full of new experiences.

Thank You to all those people who made this possible and are still striving hard to maintain this program.

About Me:

 </in/abdul-karim-ai>

 </abdulkarim018>

 \_abdulkarim\_

**Presentation By: Abdul Karim (260099)**