**📘 Next.js 13 Full Stack Course Summary**

Link: <https://youtu.be/ZVnjOPwW4ZA?si=QBbTXUgOcyfuqZbf>

**Summary**

- 🚀 Learn Next.js from basics to advanced full-stack app development with zero prior Next.js knowledge required.

- 🛠 Build a real production-grade issue tracking app featuring authentication, markdown editing, filtering, and pagination.

- 📁 Next.js file-based routing system explained with the new app router and nested routes.

- ⚡ Deep dive into server-side rendering (SSR), static site generation (SSG), and client-side rendering (CSR) with performance and SEO implications.

- 🎨 Styling covered comprehensively using global CSS, CSS modules, Tailwind CSS, and DaisyUI component library.

- 🔄 Efficient data fetching with server components, fetch caching, and TypeScript for type safety and better developer experience.

- 🎓 Instructor with 20+ years experience offers a full 5-hour course with certificate and refund guarantee for structured learning.

**Key Insights**

- 🌐 \*\*Next.js as a Full Stack Framework:\*\* Unlike React which is purely a UI library, Next.js provides a full-fledged framework integrating routing, rendering, and backend runtime under one umbrella. This simplifies development by allowing frontend and backend code to coexist seamlessly in a single project, improving maintainability and reducing complexity. The built-in Node.js runtime enables server-side rendering and API routes without maintaining separate backend services.

- ⚙️ \*\*Server vs Client Components and Rendering Models:\*\* Next.js 13 introduces server components by default, which are rendered on the server and sent as static HTML to the client. This reduces JavaScript bundle size and improves SEO since search engine bots receive fully rendered content. Client components, which handle user interactions and state, are used sparingly to optimize performance. This division allows developers to build faster, more efficient apps by minimizing client-side JavaScript and keeping sensitive logic on the server.

- 📊 \*\*Data Fetching in Server Components with Caching:\*\* Fetching data directly in server components using the native `fetch` API eliminates the need for client-side state and effects to manage asynchronous data. This approach enables Next.js to cache fetched data automatically at the filesystem level, significantly improving response times for repeated requests. Developers can control caching behavior with options like `no-store` for dynamic data or `revalidate` intervals for incremental regeneration, balancing freshness and performance effectively.

- 🧩 \*\*File-based Routing with the New App Router:\*\* The new app router in Next.js 13 uses a convention-over-configuration approach, where folder and file names map directly to routes. This simplifies routing management and supports nested routes naturally. Unlike the legacy pages router, assets inside route folders are not publicly accessible unless explicitly exposed, improving security and project organization. The new router also supports React Server Components natively, encouraging modern app architecture.

- 🎨 \*\*Styling Approaches and Best Practices:\*\* The course teaches multiple styling strategies: global CSS for truly global styles, CSS modules for component-scoped styles to avoid class conflicts, Tailwind CSS for utility-first styling that allows rapid UI development, and DaisyUI for ready-made Tailwind components. Tailwind’s utility classes enable developers to keep styles close to markup, reducing context switching and unused CSS. DaisyUI enhances productivity by providing accessible, themeable UI components built on Tailwind.

- 🛡 \*\*TypeScript Integration for Improved Developer Experience:\*\* Using TypeScript to define interfaces for data objects, such as API responses, provides autocomplete and static type checking at compile time. This reduces runtime errors and improves code quality. The instructor emphasizes annotating data types to avoid implicit `any` types, which helps catch typos and inconsistencies early, making the codebase more robust and maintainable.

- ⚡ \*\*Navigation Optimization with Client-side Transitions:\*\* Replacing traditional anchor tags with Next.js `Link` components enables client-side navigation, which avoids full page reloads. This technique only fetches the necessary content for the new page, reducing network requests and improving perceived application speed. Such smooth transitions are key to modern Single Page Application (SPA) experiences, while still leveraging server rendering benefits.

**Conclusion**

This transcript offers a thorough overview of a well-structured Next.js course that equips learners with the skills to build scalable, performant, and SEO-friendly full-stack applications. The course covers essential Next.js features, best practices for routing, rendering, data fetching, and styling, all demonstrated through practical coding examples. It emphasizes leveraging server components, caching, and TypeScript to write efficient, maintainable code. Tailwind and DaisyUI are introduced as modern styling solutions that speed up UI development while maintaining clean codebases. The instructor’s practical approach and extensive experience make this course a valuable resource for React developers aiming to master Next.js.

**Next.js 13 Tutorial (App Router with TypeScript)**

**00:00:00**

[Music] foreign welcome to the ultimate nexjs course in this course you will learn everything you need to know about nexjs from the basics to more advanced concepts so by the end of the course you'll be able to confidently build fast and scalable applications with the xjs if you have been searching for a comprehensive easy to follow well-organized and practical course that takes you from Zero to Hero this is the right next.js course for you you don't need any prior knowledge of next.js to

**00:00:29**

get started everything you need is right here so you won't need to jump back and forth between random tutorials but here's the catch unlike other courses we're not just building a dummy app we'll be building a beautiful full stack production grade app for tracking issues an app complete with all the features and UI patterns you would expect in modern applications on the home page we have this beautiful dashboard that displays the latest issues and their status and all of this data is stored in

**00:00:57**

a mySQL database we can go to the issues page we can filter issues sort them and go to different pages we can click on an issue to see more details we can assign an issue to a user so here we have full authentication and authorization we can edit an issue and here we have this beautiful markdown editor we can also delete an issue and here we get this confirmation dialog box we'll be building this application using a Cutting Edge stack next she has 13 Tailwind Radix UI Prisma react query reactbook forms Zod and more and don't

**00:01:39**

worry if some of these tools are alien to you just like my other courses I will walk you through each one explaining the what the why and the how so if you follow along you will master nexjs and we'll be able to build full stack applications with confidence a software engineer with over 20 years of experience and I've taught Millions how to code and become professional software Engineers through my YouTube channel and online school codewitmarsh.com if you are new here make sure to subscribe as I upload new

**00:02:09**

videos all the time now let's jump in and get started [Music] all right so what exactly do you need to know to take this course well to take this course you don't need any prior knowledge of next.js because I'm going to teach you everything from the ground up however you need to have basic familiarity with react and typescript because next.js is a react framework and if you don't know react you are not really the right student for this course now if you want to learn react I have a

**00:02:40**

great tutorial on my YouTube channel the link is below this video I also have a comprehensive course that goes way beyond that it teaches you everything you need to know to build modern applications with react 18 and typescript the full course is 14 hours divided into two parts so you can easily finish each part in this course you will learn how to build this beautiful application for discovering video games here we have all the common UI patterns you see in real applications like filtering sorting infinite Scrolls and

**00:03:11**

so on again in case you are interested the link is below this video welcome back to our next chess course in this section we'll be talking about the fundamentals of nexjs first I will explain what exactly next.js is and why is it so popular shortly after we'll set up our development environment and create our first next.js project from there we'll talk about some foundational Concepts such as client and server components data fetching caching as well as static and dynamic rendering this

**00:03:46**

section is a great introduction to next.js so let's jump in and get started [Music] so you might be wondering what is this next JS thing everyone is talking about and why should I bother with it well next.js is an incredibly powerful framework for building fast and search engine friendly applications it's built on top of react so everything you have learned about react is still relevant but nexjs takes web development to the next level wild react is just a library for creating interactive user interfaces

**00:04:26**

next.js is a comprehensive framework think of a framework as a collection of libraries tools and conventions that streamline application development for instance next.js includes its own routing Library so we don't need to use a separate Library like react router in terms of tooling it comes with a compiler for transforming and minifying our JavaScript code a command line interface for building and starting our application and a node.js runtime now you might wonder what exactly is a node.js runtime well there are two main

**00:05:02**

ways we can execute JavaScript code within a web browser on the client side or within a node.js runtime on the server so a node.js runtime is just a fancy term for a program that can execute JavaScript code so next JS comes with a node.js runtime and this allows us to do some really cool things the first thing is that we can do full stack development so we can write both the front end and back-end code within the same nyxjs project the backend code gets executed within the node.js runtime and the front-end code gets bundled and sent

**00:05:38**

to the client for execution within a web browser in contrast when building applications with react we have to maintain a separate backend project in a potentially different programming language this node.js runtime also allows us to render our components on the server and send their content to the client this technique is called server-side rendering or SSR and can make our applications faster and more search engine friendly we'll talk about it in detail later in the course but wait there's more with next.js we can

**00:06:12**

pre-render certain pages and components that have static data when we build our application we just render them once and serve them whenever they are needed this technique is called Static site generation and can make our applications super fast so in a nutshell next.js is a framework for building super fast and search engine friendly applications [Music] all right now let's talk about setting up our development environment to run next.js you should have node version 16.8 or higher so head over to nodejs.org and download

**00:06:52**

the latest version now in this course I will be using vs code as my editor just like my other courses you're welcome to use your preferred editor but I encourage you to use vs code because along the way I'll be sharing a lot of tips and techniques for writing code fast so you can get vs code from code.visualstudio.com now here in vs code window let's talk about the extensions I'm going to use in this course so in this panel search for es7 all right look at this extension es7 plus react Redux and react native

**00:07:29**

this extension gives us a bunch of code Snippets so we can quickly and easily generate react components the next extension is typescript is this one here JavaScript and typescript nightly and the last one is Tailwind CSS intellisense now if you have never worked with Telvin before don't worry it's super easy and I'm going to hold your hands through the entire course [Music] all right to create our first nexgs project open up a terminal window and run npx create Dash next Dash app add

**00:08:15**

you can use the latest version but in this course I'm going to use version 13.4 so I strongly recommend you to use the same version so you don't have any difficulties going through the course let's go ahead now it's asking if you want to install this package create next app version 13.4.13 let's proceed all right now it's going to ask us a bunch of questions about our new project the first question is the name of our project I'm going to use next Dash app the next question is if you want to use

**00:08:49**

typescript in this project the default answer is yes so let's press enter to accept it the next question is about using eslint which is a common code analysis tool that we can use to find common errors like syntax errors formatting issues and so on again we're going to accept the default value which is yes the next question is about using Tailwind CSS one more time you're going to accept yes the next question is about using the source directory a lot of next.js projects don't use the source

**00:09:19**

directory so I'm going to select no here the next question is about using the new app router I'm going to talk about this later in this section but very briefly in Nexus 13 we have two types of routers we have the new app router and the Legacy Pages router in this course we're going to use the new app router so let's select yes the last question is about customizing the default import aliens we're going to select no all right now it's going to install all these dependencies so you see we have

**00:09:51**

react react on next typescript and so on all right all of our dependencies are installed so now let's go into this folder and run npm run Dev this starts at development server on Port 3000 so let's control and click on this link and this confirms that our first next.js project is open running [Music] let's talk about the key files and folders in this project so at the top we have the app folder or this is also called the app router this is the container for our routing system so in next.js our router is based on the file

**00:10:41**

system so unlike react router we don't have to configure our routes and map them to our components we can simply create files and folders to represent our routes we'll talk about them in the next lesson so in the app folder we have a favorite icon we have our Global CSS file a layout file which is a basic react component that returns an HTML and body element this represents the common layout for our Pages now inside the body element we have children which is replaced by a page dynamically at

**00:11:13**

runtime depending on where the user is in our application now in this folder we also have a page file page.tsx this represents our home page now for this demo let's delete everything here and replace it with a simple markup so we're going to return a main element inside main we want to add an H1 and say hello world now back to the browser here we have fast refresh so anytime we make any changes to our typescript or css files the changes are reflected immediately now here we have a bit of

**00:11:51**

styling issue because there is a gradient a linear gradient applied to the body element so let's go to our Global CSS file down the bottom look at the Styles applied to the body element the background attribute is set to a linear gradient and that is why we have this weird style here so simply remove the background attribute and the issue goes away beautiful now I want to apply a padding here so the content is not so close to the edges of the screen so let's add padding to one REM okay that is better

**00:12:31**

so we're done with the app folder now in this project after that folder we have the public folder this is where we can put our popping assets like images in this case we have two SVG files here which are vector graphics one is next the other is verisal which is the company that has created nexjs now in the root we have a bunch of configuration files we have one for eslint another for next post CSS tailwind and typescript for the most part we don't have to touch this configuration files but if the situation

**00:13:06**

changes in the future we'll come back and revisit them thank you [Music] all right I told you that routing in xjs is based on the file system so here in the app folder we can create a new folder called users now to make this publicly accessible here we should add a page file in this folder so page dot now the extension can be JS jsx or TSX or typescript in this course we use typescript so I'm going to go with TSX now make sure to name this file correctly page in lowercase because this is one of the conventions that nexjs

**00:13:52**

looks for so the routing system in xjs is based on convention not configuration okay so here we have a page file now in this page file we should export a react component that will be rendered when the user is at this location slash users earlier we installed a very useful extension in vs code with that extension we can generate a react component using this shortcut r a f c e that is short for react Arrow function component with an export now the way I remember this is Raf C okay so that's generated beautiful now here

**00:14:32**

we have multiple cursors activated so we can rename this component to something more meaningful like users page the name we assign here it doesn't really matter in terms of routing this is just for better organization of our code okay now let's press escape to deactivate multiple cursors so back to the browser now let's go to slash users and here's our new users page beautiful now one thing you need to know about this routing system is that if you add any other files in this folder let's say

**00:15:05**

test.css this file is not going to be accessible so if you go to slash users slash test.css look we get this not found page so this is how the new app router is different from the old Pages router in the pages router if we put any files in these folders those files would be publicly accessible but this is not the case with the new router okay so let's delete this file now here we can also create nested routes so inside the users folder we can add a new folder called new and in this folder we add

**00:15:44**

a new page file so page.tsx one more time let's create a react component and we're going to call this new user page good so now we can go to users Slash new and see this new page beautiful now let's talk about navigation so we're going to go back to our home page so here we press command and P on Mac or Ctrl mp on Windows to look up files by their name if we type page we can see all our page files so the first item is our home page this is the one we are looking for now on this page let's add

**00:16:26**

an anchor so here we have an anchor we set href to slash users and give it a label like users now there is a problem with this way of implementing navigation let me show you so let's go back to our home page all right now I'm going to open up devtools here in the network tab look at all the requests sent to the server the first one is our HTML document the second one is a font the third one is a CSS file and after that we have a bunch of JavaScript files now I'm going to clear this list

**00:17:05**

look what happens when we click on the user's link let's go back to the network tab look all these resources are redownloaded this is not the optimal way to implement navigation because in the real application we probably have a navigation bar on the top the side panel on the left so as the user navigates from one page to another we don't want to reload all these repetitive parts we only want to replace the content area right this is where we use the link component in next.js so back to our code

**00:17:40**

we're going to replace this anchor with a link component that is defined in next slash link Library okay let's replace it here as well good now back to our home page once again I'm going to bring up the network Tab and clear this list now look what happens when we click on the user's link look we only have two requests and these requests are for downloading the content of the users page so we are not re-downloading a font a CSS file and a bunch of JavaScript files this is what we call client-side navigation now there

**00:18:18**

is more to navigation we have a comprehensive section about this topic later in the course this was just a basic overview [Music] in the next rest projects we have two environments where we can render our components and generate HTML markup either on the client within a web browser or on the server within a node.js runtime rendering components on the client is similar to how react applications work we refer to this technique as client-side rendering or CSR on the flip side we have server-side rendering or SSR where components are

**00:18:59**

rendered on the server so what are the differences well with client-side rendering we have to bundle all our components and send them to the client for rendering this means as our application grows so does our bundle size because it must contain all of our components now the larger the bundle the more memory we need on the client to load all these components so this approach is resource heavy the other problem is that search engine bot Parts which are machines that browse and index our websites can't view our content

**00:19:30**

because they can't execute JavaScript code so they cannot render our components like a web browser and last but not least any sensitive data we have in our components or their dependencies like API keys will be exposed to the client now if we render our components on the server we can get rid of all these problems we only send the essential components to the client and prevent our bundle from becoming unnecessarily large also because the server handles most of the rendering we need less resources on the client plus

**00:20:03**

because rendering is done on the server and we send the actual content to the client search engine Bots can view and index our pages and finally we can keep sensitive data like API keys on the server so these are all the great benefits of server-side rendering however with server-side rendering we lose interactivity so server components which are components that are rendered on the server cannot listen to browser events like Click Change submit and so on they cannot access browser apis like the local storage they cannot maintain

**00:20:38**

state or use effects these functionalities are only available in client components so in real-world applications we often use a mixture of server and client components we should default to server components and use client components only when we absolutely need them here is an example let's imagine we want to build a page to show a list of products to build this page we probably need several components like navbar sidebar product list product card pagination and footer now in standard react applications we have to

**00:21:13**

package all these components and send them to the client for rendering but in next.js we can keep all these components on the server and minimize the bundle size there is just one exception to add a product to a shopping cart we need to handle the click event of a button typically we implement this functionality in the product card component so we have to make it a client component that's one option but there is a better way we can keep this component on the server and do most of the rendering there and instead extract a

**00:21:44**

small component that only contains the add button with this change we only ship that tiny component to the client and keep everything else on the server let's see this in action back to our project in next.js all components inside the app folder are server components by default so that means all the pages we have created so far these are server components and are rendered on the server let me show you so back to the browser let's bring up the network Tab and look at the first request this is

**00:22:18**

the HTML document that we get from the backend so look we have our content here we have our hello world and the user's link this is exactly what search engine Bots see when they browse our website in contrast if we used client-side rendering which is how standard react applications work search engine Bots wouldn't be able to see our content they would see a blank page because all components all the content is rendered on the client okay so back to our project all components inside the app folder are server

**00:22:51**

components by default now if you have worked with nextjs before I should mention that the pages router doesn't support server components so going forward you should stop using it and switch to the new app router okay now let's create a new folder here called components earlier I told you that this folder is not publicly accessible unless we have a page file inside it so that means we can co-locate our project files like our components and other building blocks with our Pages we can put them next to each other and

**00:23:24**

this is perfectly fine so here in the components folder let's add a new file called Product card.tsx here we create a basic react component now earlier I told you that server components cannot have interactivity so they cannot handle browser events like Click Change and so on so that means if we add a button here and handle the click event we get a runtime error let me show you so let's pass a basic error function and log something on the console and we set the label to add to cart now let's add this component to our home

**00:24:09**

page so we go to our first page file and add our new product card component now back to the browser look we got an error saying event handlers cannot be passed to client component props if you need interactivity consider converting part of this to a client component now here we have two options one option is to make this entire component a client component so we go to the top and use the client directive so in quotes we type use client that's all we have to do with this we tell the Nexus compiler to include this file or this

**00:24:50**

component in our JavaScript bundle and that means if this component is dependent on other components those components will automatically become client components and will be included in our JavaScript bundle so we don't have to repeat this directive on every client component okay so here's one option now if you go back to the browser the error is gone but there is a better way to make our applications faster and more search engine friendly we want to render our components on the server as

**00:25:20**

much as possible and use client components only when absolutely necessary so here our product card could have some complex markup we want to render all that markup on the server and move this button to the client so I'm going to extract this button and put it inside a separate component so here in the components folder let's add a new file add to cart again we create a basic react component on the top we use the client directive and then we move this button to our new component like this okay so now we have a client

**00:26:05**

component and we're going to use this in our product card okay with this we can remove use client from this file so this component will be rendered on the server and that means where we have a client component in this case where we have this button there is going to be a hole or a slot where react will later inject our client component okay now if you go back to the home page again we don't see any errors so this is how we can create and use client and server components hey guys I hope you've been having fun

**00:26:41**

watching this tutorial I just wanted to mention that this tutorial is the first hour of my complete next JS course so after you finish this if you want to learn more I highly recommend you to enroll in the full course because it's much faster and easier than jumping between a bunch of random disconnected tutorials here on YouTube the full course is 5 hours long and teaches you everything you need to build full stack applications with nexjs it also comes with a certificate of completion and a

**00:27:10**

30-day money-back guarantee so if you're not happy ask for a full refund you get all your money back no questions asked in case you are interested the link is below this video [Music] now there are two ways we can fetch data we can fetch it on the client or on the server to fetch data on the client we typically use the state hook to declare a state variable and the effect hook to call the back end get the data and put into our state variable now in my react course I talked about react query as a

**00:27:44**

better alternative to manually using the state and effect hooks but regardless of how we fetch data fetching data on the client in client components has all the problems we talked about in the previous lesson so over time our bundles would get larger because we have to ship more and more components to the client also this approach is resource intensive because all the rendering is done on the client plus our content or our data is not visible to search engines and this approach is less secure because our API

**00:28:15**

keys or any kind of sensitive data will end up on the client but there is one extra a problem the problem is that with this approach there is always an extra round trip to the back end so when a react application loads first the browser downloads the HTML template as well as the CSS and JavaScript files from the back end then it will send an extra request to fetch data from the back end so there's always an extra round trip to the back end now we can fetch data in our server components and get rid of all these problems let me

**00:28:48**

show you in this lesson we're going to use Json placeholder to get some dummy data into our application in case you are not familiar with Json placeholder it's a fake API for getting some dummy data so head over to jsonplaceholder.typeyoucode.com on this page you will find various endpoints for getting dummy data like a list of posts comments albums photos to-do's and users in this lesson we're going to get a list of users into our application so look each user object has a bunch of

**00:29:22**

properties like ID name username and so on so back to our project let's go to the app folder and then open the users folder and then go to this page now this is a server component and in server components we can use the fetch function that you're probably familiar with this function is defined in browsers and with this we can send HTTP requests to a backend so here we pass the URL of our endpoint which we grab from here so copy and paste it now this returns a promise so we have to await it

**00:30:00**

to get the response now because we are using ubaid here we have to make this component async now this is the beauty of this approach with this approach we don't have to use a state variable we don't have to use the effect hook with zero dependencies there's no ceremony we just call fetch to get the data and all of this happens on the server you will see that in a minute so we get the response then we call response.json this also returns a promise so we'll rate it to get the data in this case our

**00:30:33**

users next we map these users to a bunch of list items exactly like how we render them in a react application so for our markup let's replace this div with a fragment here we add an H1 called users then we add an unordered list and here we map these users we grab each user now if you type user dot look we don't see anything because the compiler doesn't know about the type of these user objects so if you hover our Mouse over this constant look the type of this constant is any which means

**00:31:10**

we don't know it could be anything this is where we can use some typescript magic to improve our code so outside of this component we Define an interface or a type called user and here we say that each user has a property called ID of type number it also has name of type string and so on we could list all the properties we have here but that is not necessary for this lesson so we're going to keep things simple so with this interface we're defining the shape of our user objects okay now where we declare this constant

**00:31:49**

we annotate it with its type which is user array okay with this when we type user Dot we can see the properties of user objects so this is the benefit of using typescript with typescript we get all the completion when we are coding and also if you have any kind of Errors like typos we can cache those errors at build time before we run and deploy our application okay so here we're going to map each user to a list item first we give it a key which we set to user.id and here we render user.name that's all we had to do so our

**00:32:29**

code is simpler there is no State and effect hook here our bundle is going to be smaller because this component will be on the server now back to the browser let's go to the users page here's our users beautiful now let's bring up the network tab and refresh the page so take a look at the document that we received from the backend so all our users are rendered right here this is because rendering is happening on the server in contrast in typical react applications or wherever we use client-side rendering the browser

**00:33:05**

initially gets a blank document then it will call the backend to fetch the data and then render the content so there's always an extra round trip to the back end and our application is not search engine friendly so to fetch data whenever possible we should fetch it in server components [Music] fetching in server components has an extra benefit and that is caching what is caching well the idea of caching is to store data somewhere that is faster to access basically there are three places where we can get the data from we

**00:33:44**

can get it from the memory from the file system or from the network now as we go down this list getting data becomes slower for example getting data from the network is always slower than getting it from the file system for this reason nexjs comes with a built-in data cache so whenever we use the fetch function to get some data next.js will automatically store the result in its data cache which is based on the file system so the next time we need the same piece of data the next time we hit the same URL next.js is

**00:34:18**

not going to go to Json placeholder it's going to get the data from its data cache from the file system of course we have full control over this caching Behavior if you have data that changes frequently we can disable caching or treat data in Cache as fresh for a certain period of time let me show you so here when we call the fetch function we can pass a second argument which is an options object let me put this on a new line so we can see clearly in this object we can set cache to no store

**00:34:52**

to disable caching that's one option and this is useful if you have data that changes frequently in those situations we always want to show fresh data to our users right the other option is to keep data fresh for a certain period of time to do that instead of sending cash to no store we set next to an object in this object we can specify configuration parameters that are specific to nexjs so here we can set revalidate to Value like 10 and this means next.js is going to run a background job and get fresh data from

**00:35:30**

the backend every 10 seconds so this is how caching Works in next.js now one thing you need to know is that this caching behavior is only implemented in the fetch function so if you use a third-party Library like axis you're not going to get this you're not going to get the data cache [Music] in next ways we have another performance optimization technique called Static rendering or static site generation the idea of static rendering is that if you have Pages or components that have static data we can have an xjs render

**00:36:09**

them once when we build our application for production so next time those pages or components are needed next.js is not going to re-render them it's going to get their payload or content from its cache which is based on the file system this is static rendering meaning rendering at build time in comparison we have Dynamic rendering which happens at request time let's see this in action so back to our users page let's add a timestamp above the list of users so here we add a paragraph and render

**00:36:42**

first we create a new data object and then call to Locale time string with this we can see when this page was rendered now back in the browser let's refresh a few times so look the timestamp is changing this only happens in development mode but if you build this application for production you will see that the timestamp is not going to change because next.js will treat this page as a static page and here's the reason earlier I told you that by default whenever we use the fetch function next.js will cache the data so

**00:37:17**

it treats our data as static or unchanging data so when rendering this page next.js is that this page has static data so it decides to render this page statically at build time but if we disable caching next.js thinks the data on this page is going to change so it's not going to render this page statically it's going to render it at request time to see this action let's go back to the terminal window stop this process now earlier we used npm run Dev to start this application in our development

**00:37:50**

server now to build this for production we run npm run build alright here you can see all the routes that are generated when we build our application for production so this is our root route or the home page we have a route for our favorite icon we have one for the users page another for the users Slash new page and so on now look at the icon before each of these routes here we only see circles now down here you can see that a circle means static so these pages are automatically rendered as static HTML

**00:38:28**

now if you start this application in production which we can do by running npm start and then go back to the browser let's refresh look the timestamp is not changing because this page was rendered statically at build time now back to this page let's disable caching and rebuild this application for production so as a second argument we pass an options object and here we set cache to no store okay now back to the terminal let's stop this process and rebuild the application lovely now look at the icon

**00:39:09**

before the user's route instead of a circle we have a Lambda and down here you can see that Lambda means rendering on the server so server side renders at runtime now let's start the application in production back to the browser let's refresh now look every time we refresh the timestamp changes let's recap everything I've learned about rendering in next.js rendering can happen on the client or on the server if it happens on the server it can happen at build time which is called Static rendering or at request

**00:39:46**

time which is called Dynamic rendering so that brings us to the end of this section in the next section we'll talk about styling next.js applications so I will see you there welcome back to our next chess course in this section we'll explore various ways to style our applications and make them beautiful we'll cover Global Styles CSS modules tailwind and Daisy UI so let's jump in and get started [Music] earlier in the course we briefly talked about our Global style shade in the app

**00:40:27**

folder so let's open up globals.css in this file on the top we have three directives to import the base styles from Tailwind now we're going to talk about Tailwind later in this section but very briefly Tailwind is a very popular CSS framework okay then after that we have this root selector this is where we typically Define custom properties like foreground RGB and so on then we have media prefers color scheme dark with the CSS feature which is relatively new we can detect if the user is in the dark mode or not if so you can

**00:41:04**

see that the value of this custom properties is overwritten now down here you can see that we have used the value of this custom property as the color of the body element so we are not using background start and end RGB they were used earlier when we had a gradient on our home page so before going further let's do a bit of cleanup here I'm going to delete these two lines and these two lines from here and also let's go back to the terminal and make sure that we are running this application in

**00:41:38**

development so anytime we make any changes here we can see the changes immediately so npm run Def good now back to this file so this is our Global style shade and we should use it for styles that are truly Global in our application styles that apply to all pages like anything to do with the body element are headings our hyperlinks and so on anything that is specific to a page or a component we should implement it somewhere else so one thing you shouldn't do here is you shouldn't Define custom classes like

**00:42:12**

user list because this only applies to a particular component or a page with this approach as we build more and more components This Global style sheet grows really large and becomes unmaintainable and more importantly as we change or delete our components we'll have to remember to come back and do some cleanup here otherwise over time we'll have a lot of depth Styles in this file if you have been working with CSS for a while you know what I'm talking about so Reserve this file for styles that are

**00:42:42**

truly Global for styles that are specific to a page or a component use CSS modules or Tailwind which we're going to talk about next [Music] all right let's talk about CSS modules a CSS module is a CSS file that is scoped to a page or a component it's a way to prevent styles from clashing or overwriting each other if you have been working with CSS for a while you know that if you have the same class defined in two different places in two different style sheets these classes can override

**00:43:18**

each other depending on how we import these stylesheet files CSS modules aim to solve that problem let's see how they work so here in the components folder we have a component called Product cart now let's say we want to create some styles for this component so here we add a new file we can call this file anything but it's better to call it the same as the component so product card what matters is that the extension has to be dot module dot CSS okay now in this file or in this CSS module we can Define classes

**00:43:56**

that are scoped to our product card component so let's define a class called card and of course we can Define this class somewhere else without worrying that these classes would Clash okay so here we can apply a padding of one ram and a border of one pixel solid CCC now we go back to our component and import this time sheet so we type import styles from current folder product card dot module.css the name we assign here doesn't matter but typically we go with Styles and this will be a JavaScript object so the classes that we

**00:44:40**

Define in the CSS module will end up being properties of that object so here if we type styles we should see those properties but I don't know why intellisense is not currently working for me so I apologize for that so here we should see the classes we have defined like card and so on and that means in CSS modules we cannot use hyphens when naming these classes like card container because this is not a valid name for a JavaScript property so in these files always use the camel case notation so we remove the hyphen and

**00:45:16**

make the first letter of the second word uppercase okay now let's just simplify things we go with card back to our component so we have this object now on this div we set class name to use our curly braces and set it to Styles Dot card now back to the browser here on the home page we can see the style applied to our cart but let me show you something interesting let's right click here and inspect this in Chrome devtools here's our div look at the class name this is not the class that we created

**00:45:55**

right this is auto generated so what is happening here is that this project uses a tool called post CSS for Transforming Our CSS class names so back to our project here in the root folder we have a file called post CSS dot config.js in this file currently we have two plugins one is Tailwind the other is auto prefixer now for the most part we don't have to touch this configuration file but if you're an advanced post CSS user you know this is where you can provide your custom configuration

**00:46:30**

so when building our application next.js uses post CSS to transform our class names and generate unique class names that don't Clash this is how CSS modules work now one last thing before we finish this lesson currently I put the CSS file and the typescript file next to each other if you don't like mixing up your CSS and TSX files you can group them in folders so here we can create a folder like product card this is a container for our component in this component we can have a TSX file and a CSS file okay

**00:47:08**

and of course we can use the same technique with our pages so we can use CSS modules to Define styles that are local to a particular page [Music] all right let's talk about Tailwind so Tailwind is a very popular CSS framework that uses the concept of utility classes so here on tailwindcss.com you can see that we have a bunch of classes like Flex pt-4 which is short for padding top 4 text Center and so on so here we have a ton of small utility classes and we can combine them to style our application

**00:47:49**

some people love it some people hate it I have to convince that I also have a love-hate relationship with Tailwind but I think I love it more than I hate it if you have never used it before I strongly encourage you to give it a chance because these days a lot of projects are built with Tailwind so if you want to upgrade your skills and expand your job opportunities you should have Tailwind in your skill set so in Tailwind we have a ton of small utility classes for example for controlling paddings we have

**00:48:20**

a bunch of classes that start with P followed by a number which can be one two three four and so on the larger the number the more padding we'll have we also have PX for applying horizontal padding this is the same as applying a left and a right padding we also have py for vertical padding we have PT for padding top PR for padding right PB or padding bottom and pl for padding left now you don't have to memorize any of this stuff as you practice you'll get the hang of them pretty quickly now for

**00:48:53**

margins we have similar classes but they start with M followed by a number MX or horizontal margin and Y for vertical margin and so on now for styling text again we have a ton of classes for example for controlling the size we have text Dash XS which is extra small we have text small text base which is like the regular size we have text large x-large 2x large and so on for applying colors we have classes that start with text followed by a color so if you Google Tailwind color palette on this page

**00:49:32**

you can see the default color palette that comes with Tailwind so we have all these beautiful colors and for the most part we don't need to customize them they just look great as they are but you can always customize these and use colors that are specific to your brand Now look for example for Sky we have various shades of blue that start with 50 and go all the way to 950. the larger the number the darker the color you will see an example of this later in this video now back to our slide we're

**00:50:03**

controlling the background color we have a bunch of classes that start with BG followed by a color code now for controlling the thickness of font we have a bunch of classes like font thin font light normal medium bold and swap that's the idea of Tailwind now there are a lot more classes in Tailwind that we can cover here but our Focus here in this course is on nyxjs and not Tailwind I just want to show you the basics so you can learn the rest on your own so let's say these classes in action and

**00:50:33**

then I will explain why you may want to use Tailwind so back to our project earlier we used the CSS module to style our product card let's see how we can use Tailwind instead of a CSS module to style this component so in this file we don't need to import our CSS module instead we can apply Tailwind utility classes right here now in this lesson I'm going to use a different set of styles than the Styles we applied in our CSS module so you can see more of Tailwind in action so we're going to give this some padding

**00:51:10**

we type p dash that if you press control and space you can see all the padding classes so we have p-0 which is equivalent to padding zero we have padding one which is equivalent to 0.25 Ram or 4 pixels and as the number increases we get more padding so this intellisense comes with the extension that we installed at the beginning of the course so let's give this a padding of five a vertical margin of 5 now I want to give it a background color so we type BG Dash that look again in intellisense we can see all the colors

**00:51:49**

I'm going to go with Sky Dash 400. now to make the text white here we apply text Dash White we can make it large so here we use text Excel or 2XL 3XL and so on again on the right side you can see the actual size that is applied so for text Excel font size is 1.25 Ram or 20 pixels now here's a part that gets interesting here we have sudo selectors like hover and with this we can apply Styles when we hover over this box or this div so we can change the background color to BG sky 600 or 500 to make it slightly darker

**00:52:35**

now back to the browser here's our div if we hover over it you can see the color changes so this is how we style our applications with Tailwind now the selling point of Tailwind is that we can start our components right here in our component file we don't have to juggle back and forth between a CSS file and a component file everything is in one place now some people argue that this violates the separation of concerns principle but I have to disagree with that because the whole idea of separation of concerns is

**00:53:07**

organize our code into distinct sections or modules each having a separate concern if you follow this principle we'll have more opportunities for reuse but in this case this react component itself is a module and defines the ReUse boundary so what we have inside this boundary is purely implementation detail we can have markup with JavaScript and style all next to each other what is inside is irrelevant to the outside world as long as this component or this module is reusable that's just what I

**00:53:40**

think if you disagree or don't like using Tailwind that's totally fine but again if you want to expand your job opportunities I highly encourage you to learn Tailwind because it's in high demand these days but there is one more benefit to using tailwind and this is the reason I personally love Tailwind with Tailwind when we build our application our final CSS bundle will only have the utility classes that we have used in our markup so if tomorrow we delete this div none of these classes

**00:54:08**

will be in our final CSS bundle unless we have used them somewhere else so with Tailwind we don't have to remember to clean up as we change or delete our components with CSS modules if we delete this component we have to remember to delete this CSS file as well or if we change the structure here we might still have unused classes in our CSS module so we have to remember to come back and clean up so that's the selling point that convinced me to start using Tailwind but on the flip side I don't

**00:54:39**

like that over time as our marker gets complex will have a lot of these classes and the code is kind of hard to read but it's just a matter of getting used to it it's not terribly bad but it's something that puts some people off early on but over time I think the benefits outweigh the downside all right now let's talk about Daisy UI Daisy UI is a very popular component library for Tailwind it's kind of like bootstrap for Tailwind so if you head over to daisyui.com you can see all

**00:55:16**

these components for example we have accordions alerts we have breadcrumbs buttons cards Carousel chat Bubbles and so on very very useful it's very easy to use so let's go to the installation page first we have to install Daisy as a development dependency so I'm going to copy this line now back in vs code I've opened my terminal window you can open it from terminal new terminal now let's paste that good next we should add daisy as one of the plugins of Tailwind so we should go to

**00:56:00**

Tailwind configuration file in the root of our project so that is right here currently there are no plugins so let's copy this line and paste it here there's literally all we have to do now let's see how we can use these components so as an example we're going to look at the button component so here we have a bunch of classes like BTN BTN neutral BTN primary secondary and so on exactly like bootstrap under the hood these classes use Tailwind so instead of us manually combining a bunch

**00:56:42**

of small Tailwind classes to create a button we can just use the button that comes with Daisy and of course we can always customize these buttons okay so to create a button look we should have an example down here there you go this is just a basic button we also have neutral primary secondary and so on so back to our code less style this component that we created earlier add to cart so here we have a button we set class name to BTN BTN Dash primary we don't have to import anything in this file okay now back to the

**00:57:21**

homepage here's what we get let's remove this blue background because it looks kind of odd so we're going to go back to product card the TSX and I'm going to remove all these Tailwind classes okay so here's what we are left with now in DayZ we have the concept of themes for example if you look at this page themes you can see all the available themes now in this lesson I'm going to use a theme called winter and you can always preview these themes so on the top you can select them and see what your

**00:58:00**

application looks like here's the dark mode here we have the cupcake theme and so on now to use a theme there are two steps we have to follow first we have to go in Tailwind configuration file and add this section for days UI so here we activate the themes we want to use so let's copy this section and go back to Tailwind config.ts right after plugins we paste this so we have Daisy UI we have themes I don't want to use any of these I only want to use the winter theme the second step is to

**00:58:38**

apply this theme on our HTML element so back in the documentation look here we have a data attribute for specifying the theme to do that we have to go to our layout file because this is where we have our HTML element so we set data Dash theme to Winter now back to our home page our button looks blue beautiful now let's go to the users page and put these users inside a table so let's go to the users page and replace this unordered list with a table so we select this and press command and D on

**00:59:18**

Mac or Ctrl ND on Windows to activate multiple cursors now we can change these two elements in one go so we change it to a table then we press escape to deactivate multiple cursors now inside this table we want to have a t head inside the T head we want to have a TR and inside the TR we want to have two th elements let's press tab here's our markup beautiful now here we want to have two columns the first one is username the second is user email now after the t-hat element we should add a tea body

**00:59:56**

then we move mapping our users inside T body and replace Li with TR so once again we press command and D on Mac or Ctrl and D on Windows to change both these elements in one go so let's change it to tr the inside is TR we want to have two th elements so th times two let me put it on a separate line good in the first th we want to render user.name in the second moon render user dot email and currently we don't have email in the interface that we defined for our users that is why we get a compilation error so this is another

01:00:40

benefit of using typescript the typescript compiler tells us about these issues before we run or deploy our application we can catch these errors at compile time or even while coding so up here we have the user interface let's add an email property of type string now the error goes away lovely so back to the browser here's what we get now let's apply a couple of classes from Daisy to make this table look a little bit nicer so back to our code here's our table we set class name to table and table

01:01:19

Dash bordered and of course you can find these in the documentation for the table component in DayZ UI okay this is much better now I made a mistake earlier when rendering these rows I used the th element that is why they appear as bold but we should use th only as the column of our tables so let's change all of these th elements to TD once again we select one of them and we press command and D to select more of them so now we have four cursors let's change all of them to t d okay here's what we get now we no longer

01:02:01

need the timestamp so let's remove it from here as well good so we're done for now in the next section we're going to talk about routing and navigation in more detail hey guys I hope you've been having fun watching this tutorial I just wanted to mention that this tutorial is the first hour of my complete next JS course so after you finish this if you want to learn more I highly recommend you to enter on the full course because it's much faster and easier than jumping between a bunch of random disconnected

01:02:33

tutorials here on YouTube the full course is 5 hours long and teaches you everything you need to build full stack applications with nexjs it also comes with a certificate of completion and a 30-day money-back guarantee so if you're not happy ask for a full refund you get all your money back no questions asked in case you are interested the link is below this video