NexJs:

Framework of react

Extensions:

* ES7+ React/Redux/React-Native (snippets of react components that do development faster) Rafce
* JavaScript and TypeScript Nightly
* Tailwind CSS intellisense

For Creation Of app:

npx [create-next-app@13.4](mailto:create-next-app@13.4)

For running the app:

Npm run dev

Project Structure:

1)app folder:

It is also called app router,it is a container for our routing system our router is based on the file system unlike react router we don’t have to configure and map them to components we simply create files and folders to represent our routes

* Layout.tsx:

it is a basic react component that returns an html and body element

* Page.tsx:

Main page

2)public folder:

Here we put public assests like images

Root Directory:

1)Eslintr.json

2)next.config.js

Routing And Navigation:

Users folder in app folder

Nested routing:

users->new

In Home Page we should not do navigation by anchor tag due to this all other documents redownloaded we have to use Link tag for navigation and this is called client side navigation

Difference Between Client Side Rendering And Server Side Rendering:

Client Side Rendering:

1)Large Bundles

2)Resource Intensive

3)No SEO

4)Less Secure

Server Side Rendering:

Smaller Bundles

Resource Efficient

SEO

More Secure

Server Components cannot listen to browser events,access browser APIs,Maintain state,Use Effectws

In next js all the components are server components by default.

Error:

**Unhandled Runtime Error**

**Error: Event handlers cannot be passed to Client Component props. <button onClick={function} children=...> ^^^^^^^^^^ If you need interactivity, consider converting part of this to a Client Component.**

import Image from 'next/image'

import Link from 'next/link'

import ProductCard from './components/ProductCard'

export default function Home() {

  return (

   <main>

    <h1>

      Hello World

    </h1>

    <Link href="/users">

    Users

    </Link>

    <ProductCard/>

   </main>

  )

}

import React from 'react'

const ProductCard = () => {

  return (

<div>

    <button onClick={()=>console.log('Click')}>

    AddToCart

    </button>

</div>

  )

}

export default ProductCard

solution:

1)úse client’ on top of the component:

With this we tell the next js compiler to include this file or component in our javascript bundle

And that means if this component is dependent upon other components those components will

Automtically cecomes cleint components and will be included in javascript bundle

2)

For making application faster and seo friendly

For this we have to do server side rendering as much as possible

Here out product card have some complex markup we want to render all markup on server and

Move product card addtocart button to the cleint so I want to extract that button and move it into new file

'use client'

import React from 'react'

const AddToCart = () => {

  return (

    <div>

     <button onClick={()=>console.log('Click')}>

    AddToCart

    </button>

    </div>

  )

}

export default AddToCart

import React from 'react'

import AddToCart from './AddToCart'

const ProductCard = () => {

  return (

<div>

   <AddToCart/>

</div>

  )

}

export default ProductCard

Data Fetching:

1)Fetch Data on client

2)Fetch Data on server

Fetching Data on Cleint:

We use UseState() hoook to declare a state variable and 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 better alternative to manually using the state and effect hooks

Problems:

1)Large Bundles

2)Resource Intensive

3)No SEO

4)Less Secure

5)Extra roundtrip to server:

So whena react application loads first bbrowser downloads the HTML Tempalte as well as the CSS and javascript files from the back end then it will send an extra request to fetch data from the backend

Solution:

We can fetch data In our server component

Caching:

Fetcing in server components has an extra benefit and that s caching

Idea of caching is To store data somewhere to faster to access basically there are three places from there we can get the data from:

Data Sources:

1)Memory

2)File System

3)Network

As we go down in the 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 next js comes with the built in data cache whenever we use the fetch function to get some data it automtically stores 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 url next js is not going to go to Json placeholder it’s going to get the data in the cache from the file system of course we have full control over this caching behaviour if you havee data that is change frequently you can disable caching or treat data in cache as fresh for a certain period of time

To show fresh data to user:

Options:

1)

const UsersPage = async () => {

    const res=await fetch('https://jsonplaceholder.typicode.com/users',

    {cache:'no-store'}

    );

2) revalidate after 10 seconds

const UsersPage = async () => {

    const res=await fetch('https://jsonplaceholder.typicode.com/users',

    {next:{revalidate:10}}

    );

In third party library axios you can’’t get data cache

Static Rendering Or Static Side Genereation:

If you have pages or components that have static data we can have next js render them once when we build our application for production so next time those pages or components that needed next js is not gonna rerender them it will get payload and content from their cache which is based on file system that is static rendering meaning rendering at build time.

Dynamic Rendering:

Dynamic rendering which happpens at request time let’s see this 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 first we create

A new data object and then call to locale time string

Npm run build-> to see routes and if it is in production mode time changes works

Statically and if npm run dev which is development mode works dynamically

For dynamic work on production write cache:no-store

Styling Next Js Applicatons:

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 iknkow that if you have the same class defined in two different places in two different style sheets these classes can override each other depending on how we import hese stylesheet fiels CSS css MODULES aim to solve that problem

Postcss.config.js:

Autoprefixer plugi autogenerate class name of css element

Tailwind CSS:

It uses the concept of utility classes

Spacing:

Paddings:

p-[number]

px – [number] ->hortizational

py –[number]->vertical

pt –[number]->padding top

pr-[number]->padding right

pb-[number]->padding bottom

pl-[number]->padding left

Margings:

We have similar classes but start with m

Text:

Size:

text-xs->extra small

text-sm->small

text-base->regular size

text-lg->

text-xl->

text-2xl->

text-3xl->

Color:Tailwind Color pallete google

text-[color]

bg-[color]

Thickness:

font-thin

font-light

font-normal

font-medium

font-bold

tailwind css:

we don’t have to write code of css in separate file for tailwind

<div className='p-5 my-5 bg-sky-400 text-white text-xl'>

   <AddToCart/>

</div>

Daisy UI:

It’s very popular component library for tailwind it’s gonna like bootstrap for tailwind

Portfolio Website:

Npm install react-icons