* npm create next-app@latest <nombre proyecto>
* npx create-next-app@latest

# 17 inferior

## /pages

Aquí se van a ubicar las rutas (es como sveltkit),

Al nombrar rutas evitar usar el ‘-’

Si se crea un archivo cuyo nombre sea distinto des ‘index.jsx’, se considera una ruta a partir de esta carpeta /miruta

## /pages/MiRuta/index.jsx

Si se crea un carpeta y dentro un index.jsx entonces esto es una nueva ruta /Miruta donde se renderiza el contenido de index.jsx

export default function index(){

    return (

        <>

        Hello

        </>

    )

}

# /app/page.tsx .jsx

Es la primera vista del proyecto

Cada nueva carpeta que se cree que tenga un archivo page.tsx o .jsx, se considera una ruta

Y debería de tener una función componente llamada Page, aunque creo que no es ncesario este nombre

export default function Page(){

    return (<>

        <h2>Mi ruta1</h2>

    </>)

}

# /app/layout.tsx .jsx

Tiene el componente layout por defecto

Envuelve a sus hijos según la carpeta en la que este

export default function Layout(

    {children}: {children:React.ReactNode}

){

    return(<>

        Mi Layout

        {children}

        End Mi Layout

    </>)

}

# /app/lib

Archivos js con utilidades

# /public/

Poner los archivos estáticos, como el /img/

# Link

Import Link from ‘next/link’

<link href=’/ruta/mivista’ > Algo </>

# Image

Import Image from ‘next/image’;

<Image src=”..jpg” width={600} height={600} alt=”descripcion” />

La carpeta public se considera la raíz asi que no hay que incluirla

# Metadatos

Para el CEO

Import Head from “next/head”;

<Head>

<title> Mi titulo</>

<link rel=“icon” ../>

<meta …/>

</Head>

# Layout

Crear un componente (en cualquier parte, pero preferiblemente en su carpeta components) que haga de esta función,

Export default funtion MiLayout({children}){

Return (

…

{children}

)

}

Tener en cuenta que en la ruta que se declaren va a ser utilizado tambien en sus subrutas (creo?)

# defaultProps

Para poner parámetros por defecto tambien se puede

Componente.defaultProps={

    title:"el titulo"

}

# Estilos en modo clase

/styles/miestilos.css

.container{

    max-width: 36rem;

    font-size: large;

    color: blue;

}

En el componente

import styles from '../styles/Layout.module.css'

<div className={styles.container}>

En los estilos no se pueden usar etiquetas html, solo clases

Fuentes de Google

/app/ui/fonts.tsx

Cargar en un archivos las fuentes de Google que trae Next

import { Inter,Montserrat} from 'next/font/google';

export const moneserrat= Montserrat({subsets:['latin']})

Luego usarlas en otro componente

import {moneserrat} from './ui/fonts'

<h1

           className={`${moneserrat.className} antialiased`}

          >

          Hola</h1>

# getStaticProps

Supuestamente este metodo no se compila al frontend

export default function index({data}){

    return (

        <>

        Hello

        {

            data.map(({id,title,body}, index) => {

                return (

                    <div key={id}>

                        <h3>{id} - {title}</h3>

                        <p>{body}</p>

                    </div>)

        })

        }

        </>

    )

}

export async function getStaticProps() {

    try{

        const response = await fetch('https://jsonplaceholder.typicode.com/posts')

        const data = await response.json()

        return {

            props: { data },

            revalidate: 60 // 1 minute

            }

    }catch(error){

        console.error(error)

    }

}

# Rutas dinámicas

/pages/ruta/[parametro].jsx

Parecido a sveltekit

## params

Se puede acceder a los parámetros de la ruta mediante el atributo params que recibe el Page

export default function Page({params}:{

    params:{productId:string}

}) {

    return (<>

        <p>El id es: {params.productId}</p>

    </>)

}

# Estilos globales

En un layout

import "./globals.css";

# use client

para hacer se renderice en el cliente

'use client'

import { usePathname } from "next/navigation"

export default function Page(){

    const pathname=usePathname()

    return (<>

        <h2>Mi ruta2 {pathname}</h2>

    </>)

}

# Postgres

* pnpm install @vercel/postgres

# Seed

Suponiendo que hay un archivo /scripts/seed.js y tienes un .env

Agregar al package.json un script de arranque para esto

"scripts": {

    "dev": "next dev",

    "build": "next build",

    "start": "next start",

    "lint": "next lint",

    "seed": "node -r dotenv/config ./scripts/seed.js"

  },

Luego

* npm run seed

# async Componentes

Los componentes de react que se renderizan del lado del servidor pueden ser async, incluyendo los Page

export default async function Page(){

    return (<>Es asincorono</>)

}

## Loading

En lo que carga un componente se puede crear otro que se muestre mientras el usuario espera, para eso al mismo nivel del page.tsx se crea un loading.tsx

export default function Loading(){

    return (<>Cargando...</>)

}

# Suspense

Si se tiene un componente asíncrono que carga lento, se puede usar de react el Suspense para crear el efecto de cargando.., este suspense recibe un argumento fallback donde se le pasa el componente de cargando

Componente lento

export default async function ComponenteLento(){

    await new Promise((resolve)=> setTimeout(resolve,3000));

    return (<>Lento</>)

}

Se utiliza

import ComponenteLento from "@/app/lib/components/component1"

import { Suspense } from "react"

export default function Page(){

    return (

    <>

    Lento2

    <Suspense fallback={<>Espere...</>} >

        <ComponenteLento/>

    </Suspense>

    Lento2

    </>

)

}

## Key

El suspense solo carga una vez, pero si se necesita que se actualice cada ves que cambie algo específico se le agrega una key string dinámica

import { Suspense } from "react";

import Search from "../lib/components/search";

import ComponenteLento from "../lib/components/component1";

export default  function Page(

    {searchParams}

    :{

        searchParams?:{

            query?:string

            page?:string

        }

    }){

        const currentPage= Number(searchParams?.page)||1

        const query= searchParams?.query || ''

    return (

    <>Busqueda: <Search placeholder="termino"/>

        <p><span>Page:</span>{currentPage}</p>

        <p><span>Query:</span>{query}</p>

        <Suspense

        key={query+currentPage}

        fallback={<>Espere...</>} >

            <ComponenteLento/>

        </Suspense>

    </>)

}

# Skeleton

Crearlo en skeletonreact.com

# Estado en url

La idea es que si un usuario copia su url y se la lleva a otro navegador, vea lo mismo

## Search

En caso de tener un iput de búsqueda, que afecta la url, podemos sincronizarlo con los Query params de la url

Componente search

"use client";

import { useSearchParams, usePathname, useRouter } from "next/navigation";

export default function Search({placeholder}:{placeholder:string}){

    const searchParams= useSearchParams()

    const pathname=usePathname()

    const {replace} =useRouter()

    const handleSearch =(term:string)=>{

        const params=new URLSearchParams(searchParams);

        if(term){

            params.set('query',term)

        }else{

            params.delete('query')

        }

        replace(`${pathname}?${params.toString()}`)

    }

    return (<>

        <label htmlFor="search">Buscar</label>

        <input

        style={{color:"red"}}

            onChange={(event)=>handleSearch(event.target.value)}

            defaultValue={searchParams.get('query')?.toString()}

        />

    </>)

}

searchParams

Los page reciben automáticamente los Query params, como un parámetro

export default  function Page(

    {searchParams}

    :{

        searchParams?:{

            query?:string

            page?:string

        }

    }){

    return (<>Busqueda: <Search placeholder="termino"/></>)

}

# Use-debounce

Para cuando tienes un input que va a hacer algun tipo de consulta cada ves que cambie algo y, la idea para no sobrecargar de consultas es hacer una consulta cada ciertos cambios

* pnpm install use-debounce

import {useDebounceCallback} from ‘use-debounce’

const handleSearch = useDebounceCallback(

función de búsqueda , 300//cantidad de milisegundos que espera a ver si termino de escribir

)

# Acciones

Se suelen poner en app/lib/actions.ts

La primera línea debe de ser 'use server', con lo que se marca el archivo para que todas las funciones que se exportan en este archivo son de servidor y por lo tanto no se ejecutan ni se envían al cliente

'use server'

export async function createInovece(formData:FormData) {

    console.log(formData)

}

Un componente con form

import { createInvoice } from "../actions";

export default function FormExapmple(){

    return (

        <div className="form" style={{color:'red'}}>

            <form action={createInvoice}>

                <label>

                    Name:

                </label>

                <input type="text" name="name" />

                <label>

                    Age:

                </label>

                <input type="number" name="age" />

                <button type="submit">Submit</button>

            </form>

        </div>

    )

}

# Zod

* pnpm install zod

'use server'

import {z} from 'zod'

const CreateInvoiceSchema= z.object({

    id:z.string(),

    invoice\_number:z.coerce.number(),

    customer\_name:z.coerce.string()

})

const CreateInvoiceFormSchema= CreateInvoiceSchema.omit({

    id: true

})

# revalidatePath

Manda a recargar una ruta

import { revalidatePath } from 'next/cache'

revalidatePath('/rutaform')

# redirect

import { redirect } from 'next/dist/server/api-utils'

redirect('/rutaform')

# not-found

Para sobrescribir la Plantilla del error 404 crear el archivo app/not-found.tsx, como esta a este nivel hereda el layout

export default async function NotFound(){

    return (<>No lo encuentro</>)

}

# Ocultar rutas

Al nombre de la carpeta se le agrega un “\_” delante del nombre “\_oculto”