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

# 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

# Layout

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

export default function Layout(

    {children}:{children:React.ReactNode}

)

{

    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>

# Estilos globales

En un layout

import "./globals.css";

# 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>

    </>)

}

# 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

# useRouter

solo se puede usar del lado del cliente

## redirigir

"use client";

import { useRouter } from "next/router"

export default function Page() {

    const router =useRouter();

    const handlerClick=()=>{

        router.push("/otraruta")

    }

    return (<>

        <button onClick={handlerClick}>Redirigir</button>

    </>)

}

# 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 input 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”

# Agrupar rutas

Si el nombre de un carpeta se pone entre paréntesis, entonces esta carpeta como ruta es ignorada pero sus hijas no,

Ejemplo:

carpetas

/app/(agrupada)/ruta1

/app/(agrupada)/ruta2

/app/ruta3

urls

…./ruta1

…./ruta2

…./ruta3

# Metadata

En el layout principal app/layout.tsx o en cualquier otro se declaran los metadatas como una variable externa

export const metadata: Metadata = {

  title: "Create Next App",

  description: "Generated by create next app",

};

No se puede utilizar en componentes “use client”

## Metadatos

Para el CEO

Import Head from “next/head”;

<Head>

<title> Mi titulo</>

<link rel=“icon” ../>

<meta …/>

</Head>

## Dinámicos

Para agregárselos de forma dinámica por ruta, en su layout correspondiente hay que agregarle el metodo generateMetadata

import { Metadata } from "next"

type Props={

    params:{productId:string}

}

export const generateMetadata=({params}:Props):Metadata=>{

    return {

        title: `Producto ${params.productId}`

    }

}

export default function Page({params}:Props) {

    return (<>

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

    </>)

}

## Template

El template sirve para los títulos dinámicos, osea que si en una pagina ya hay un titulo definido entonces se utiliza el template superior y se sustituye el %s por el titulo de la pagina

Layout superior

export const metadata: Metadata = {

  title: {

    default:"Por defecto",

    absolute:"",

    template:"%s Resto del titulo"

  },

  description: "Generated by create next app",

};

Layout hijo

export const metadata: Metadata = {

  title: "Create Next App",

  description: "Generated by create next app",

};

El titulo resultante seria "Create Next App Resto del titulo ",

## Absolute

Para evitar que el tamplate haga efecto en un hijo hay que utilizar el absolute en el layout hijo

export const metadata: Metadata = {

  title: {

    absolute:"Este es todo el titulo",

  },

  description: "Generated by create next app",

};

El titulo de esta pagina seria :"Este es todo el titulo",

# Template

Si hay un estado en el layout, al cambiar entre sus paginas el estado se conserva

Para que esto no ocurra, osea que entre cada navegación el estado se reinicie, en ves de llamar al archivo layout.tsx seria template.tsx

# refresh

import { useRouter } from "next/navigation"

const router=useRouter()

router.refresh()

# @carpeta

Para paginas complejas donde se va a trabajar cada parte con sus propios archivos, pero dentro de la misma carpeta de la ruta, (osea si se usaran componentes estos podrían estar externos en otra carpeta como lib, pero en este caso se asume que almenos la estructura general se va a encontrar dentro de la misma carpeta )

/ruta/layout.tsx

export default function Layout(

    {children,chat,video}: {children:React.ReactNode,

        chat:React.ReactNode,

        video:React.ReactNode}

){

    return(<>

        Mi Layout Complejo

        {chat}

        -----------

        {children}

        ---------

        {video}

        Mi Layout Complejo

    </>)

}

[/ruta/@chat/page.tsx](mailto:/ruta/@chat/page.tsx)

export default function Page(){

    return (<><p>El chat2</p></>)

}

[/ruta/@video/page.tsx](mailto:/ruta/@video/page.tsx)

export default function Page(){

    return (<><p>333El video</p></>)

}

/ruta/page.tsx

export default function Page(){

    return (<>colmplejo</>)

}

Las @carpeta no son rutas, solo se cargan dentro del layout

# Api

Generalmente para ello se crea un carpeta /app/api/

Y en cada ruta carpeta un archivo /app/api/ruta/route.ts con una funcion con el nombre del tipo de metodo

export async function GET(){

    return new Response("hola mundo")

}

## Json

import { products } from "./data";

export async function GET() {

    return Response.json(products)

}

export function DELETE(){

    return NextResponse.json({

        message:"deleting single note"

        })

}

export function PUT(){

    return NextResponse.json({

        message:"updating single note"

        })

}

## Post

export async function POST(request:Request) {

    const product=await request.json();

    products.push(product);

    return Response.json(products,{

        status:201,

        headers:{

            "Content-Type":"application/json"

        }

    })

}

## error

export async function GET(){

    try {

        const notes =await prisma.note.findMany();

        return NextResponse.json(notes);

    } catch (error) {

        if(error instanceof Error){

            return NextResponse.json(error,{

                status:500

            });

        }

    }

}

# getServerSideProps

1. getServerSideProps() debe ser una función exportada.
2. getServerSideProps() solo puede ser usado en páginas (dentro de la carpeta 'pages'). No puede ser usado en componentes.
3. getServerSideProps() se debe exportar como una función independiente; no funcionará si se agrega getServerSideProps como una propiedad del componente de la página.

# Prisma

Es un ORM

* pnpm install prisma -D

crear configuraciones para las conexiones de forma automática

para postgres es por defecto

* npx prisma init

para sqlite

* npx prisma init --datasource-provider sqlite

Necesita

* pnpm install @prisma/client

## .env

DATABASE\_URL="file:./dev.db"

## /prisma/schema.prisma

Datos sobre la conexión a la base de datos

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "sqlite"

url = env("DATABASE\_URL")

}

### Modelos

En este archivo /prisma/schema.prisma tambien se pueden declarar los modelos

Ejemplo

model Note{

id Int @id @default(autoincrement())

title String

content String?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

}

Básicamente

Todo separado por 1 espacio y por saltos de línea

model NombreModelo{

nombreAtributo1 Tipo @modificador1 @modificador2(metodoArgumento())

nombreAtributo2Opcional TipoOpcional? @modificador

}

#### Tipos

Int

String

DateTime

##### Opcional ?

Si el campo puede ser nuleable se le agrega a continuación del tipo el “?”

model NombreModelo{

content String?

}

#### Modificadores

##### @id

para cuando es id

model NombreModelo{

id Int @id @default(autoincrement())

}

Suele ser acompañado de @default(autoincrement())

##### @default()

###### @default(now())

Suele ser usado para la creación de fechas de forma automática

model NombreModelo{

createdAt DateTime @default(now())

}

##### @updatedAt

Suele ser para la actualización de fecha cada ves que ocurra la creación y edición del modelo

model NombreModelo{

updatedAt DateTime @updatedAt

}

## Migraciones

La inicial y cada ves que haces un cambio

* npx prisma migrate dev --name init

Si clonaste el proyecto y vas a crear la bd y aplicar las migraciones existentes

* npx prisma migrate dev

si por algun motivo no se pudo instalar completo

* pnpm prisma generate

## studio

Supuestamente es como una administración de la base de datos, pero parece que necesita internet para funcionar

* pnpm prisma studio

Para ejecutarlo sin internet agrega al env

PRISMA\_ENGINES\_CHECKSUM\_IGNORE\_MISSING=1

## queris

### src/libs/prisma.ts

es necesario un archivo src/libs/prisma.ts donde se cree el objeto prisma global

import { PrismaClient } from "@prisma/client";

declare global {

    var prisma: PrismaClient | undefined;

}

export const prisma= global.prisma || new PrismaClient();

if (process.env.NODE !== "production"){global.prisma= prisma;}

donde se vaya a usar

import {prisma} from "@/libs/prisma"

los métodos de la api [GET,POST …] deben de ser async

### findMany

retorna todos

const notes =await prisma.note.findMany()

### create

export async function POST(request:Request){

    const {title,content} = await request.json();

    const newNote =await prisma.note.create({

        data:{

            title,

            content,

        },

    })

    return NextResponse.json(newNote);

### findFirst

interface Params{params:{id:string}};

export async function GET(request:Request,{params}:Params){

    try {

        const note =await prisma.note.findFirst({

            where:{id:Number(params.id)}

        });

        return NextResponse.json(note);

    } catch (error) {

        if(error instanceof Error){

            return NextResponse.json(error,{

                status:500

            });

        }

    }

}

### Delete

export async function DELETE(request:Request,{params}:Params){

    try{

        const deleteNote=await prisma.note.delete({

            where:{id:Number(params.id)}

        });

        if(!deleteNote) return NextResponse.json({

            message:"not found"

            },{

                status:404

            })

            return NextResponse.json(deleteNote);

    } catch (error) {

        if(error instanceof Prisma.PrismaClientKnownRequestError){

            if(error.code==="P2025"){

                return NextResponse.json({

                    message:"not found"

                    },{

                        status:404

                    })

            }

        }

        if(error instanceof Error){

            return NextResponse.json(error,{

                status:500

            });

        }

    }

    return NextResponse.json({

        message:"deleting single note"

        })

}

### Update

export async function PUT(request:Request,{params}:Params){

    try {

        const {title,content} = await request.json();

        const note =await prisma.note.update({

            where:{id:Number(params.id)},data:{

                title,

                content

            }

        });

        if(!note) return NextResponse.json({

            message:"not found"

            },{

                status:404

            })

        return NextResponse.json(note);

    } catch (error) {

        if(error instanceof Prisma.PrismaClientKnownRequestError){

            if(error.code==="P2025"){

                return NextResponse.json({

                    message:"not found"

                    },{

                        status:404

                    })

            }

        }

        if(error instanceof Error){

            return NextResponse.json(error,{

                status:500

            });

        }

    }

}

# Mongosee

* pnpm install mongodb
* pnpm install mongoose

-1ro configurar el entorno, apuntar al servidor de la base de datos, no a la base de datos

.env

MONGODB\_URI="mongodb://localhost:27017 "

-2do configurar la conexón

import mongoose from "mongoose";

const MONGODB\_URI=process.env.MONGODB\_URI;

const connect = async ()=>{

    const connectionState=mongoose.connection.readyState;

    if(connectionState===1){

        console.log("already connected");

        return;

    }

    if(connectionState===2){

        console.log("Connecting...");

        return;

    }

    try{

        await mongoose.connect(MONGODB\_URI!,{

            dbName:'testusers',

            bufferCommands:true

        });

        console.log("connected")

    }catch(err){

        console.log('error')

        console.log(err)

        throw new Error("Error",err)

    }

}

export default connect;

-3ro crear un esquema de un modelo (lo crea automáticamente en la base de datos)

import { Schema,model,models } from "mongoose";

import { unique } from "next/dist/build/utils";

const UserSchema=new Schema({

    email:{type:"string",require:true,unique:true},

    username:{type:"string",require:true,unique:true},

    password:{type:"string",require:true},

},{

    timestamps:true

})

const User=models.User || model("User",UserSchema);

export default User;