DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

Programación Web

Integrantes: Theo Rosero, Iza Marco,

Kevin Chuquimarca, Ismael Cedillo

Docente:

ANGEL MARCELO REA GUAMAN

NRC:

7175

Octubre 2021-Marzo 2022

1. **Definir el alcance del proyecto del Desarrollo WEB**

El presente trabajo es un blog que consta con un login y registro que permiten al usuario registrarse e iniciar secion , también consta de la capacidad de publicar POST los cuales podrán ser etiquetados y categorizados de acuerdo a la información de las misma , conforme al juicio del usuario al momento de publicar.

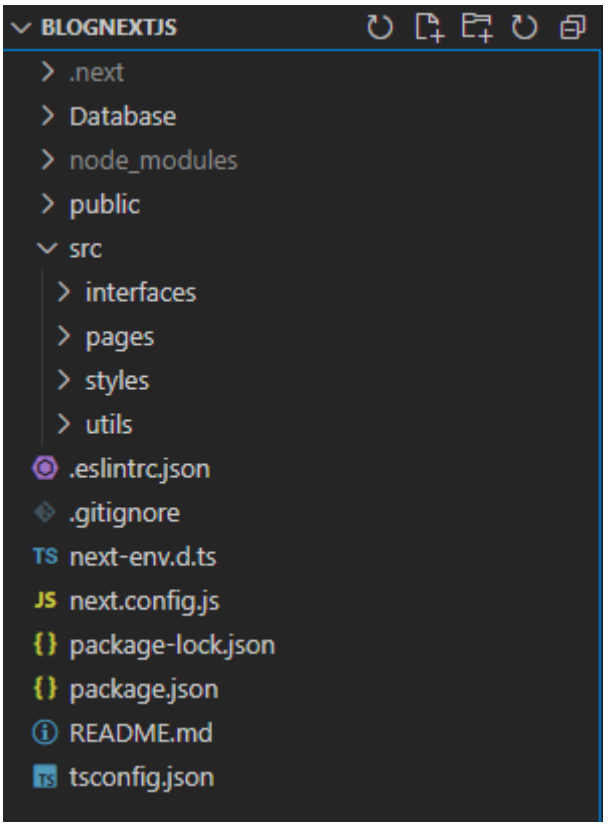
2. **Actividades realizadas**

Instalar los módulos de Next.js

```
npx create-next-app@latest --typescript
# or
yarn create next-app --typescript
```

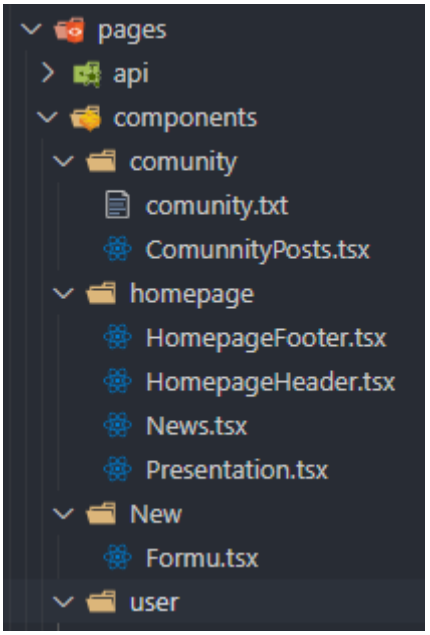
Para iniciar un proyecto con el framework Next.js se debe ejecutar el comando por terminal, ya sea usando el administrador de paquetes npm o yarn especificando que se hará uso de typescript tanto para la funcionalidad como para la creación de los componentes de react.

Definir una estructura para el proyecto



La estructura predeterminada que ofrece Next.js es modificada, incluyendo algunas subcarpetas tanto para los estilos, utilidades, interfaces, componentes, entre otras subcarpetas para organizar todos los recursos necesarios para el proyecto.

Definir los componentes de react

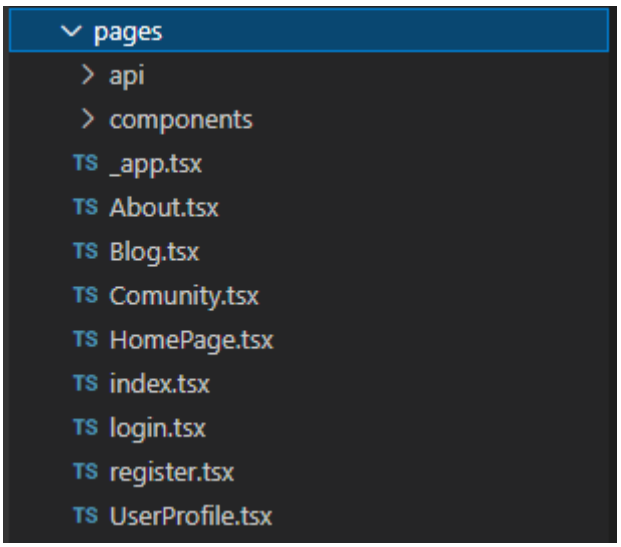


Next.js está desarrollado para ser un framework de react, por lo tanto, la gran mayoría de conceptos y funciones de la librería de react se encuentran integrados en Next.js. Para la definición de los componentes se define una carpeta llamada “componentes”, dividida en subcarpetas para una mejor organización de los componentes.

```
function Presentation() {
    return (
        <div id="presentation">
            <Image id="presentation-img" src={portada} alt=""/>
            <div id="presentation__item">
                <Image id="presentation-logo" src={logo} alt=""/>
                <Link href="/login"><a className="presentation__link">Saber Más</a></Link>
                <Link href="/register"><a className="presentation__link">Regístrarse</a></Link>
            </div>
        </div>
    );
}
```

Para definir los componentes se utilizó los archivos .tsx, estos archivos contienen componentes de react definidos con lenguaje de programación typescript, y para su comunicación y administración se hizo uso de los Hooks de react.

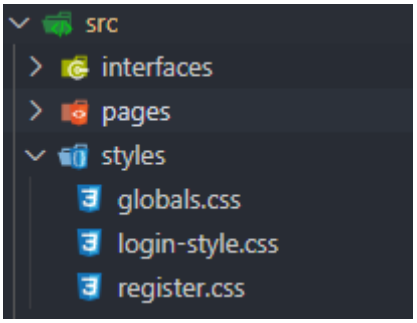
Estructuras las páginas con los componentes de react



Una vez definidos los componentes de react, se comienza a estructurar las páginas importando los componentes que sean necesarios, aquí se presenta el concepto de react y sus componentes, el poder importarlos y utilizarlos cuando sea necesario, evitando repetir código de maquetado HTML, de estilos CSS y de funcionalidad JS o typescript.

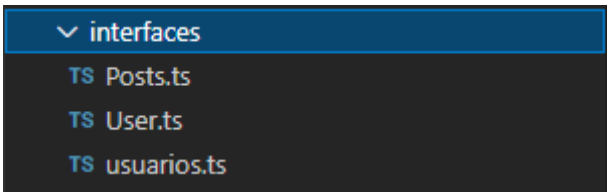
```
import HomepageHeader from './components/homepage/HomepageHeader';
import Presentation from './components/homepage/Presentation';
import News from './components/homepage/News';
import HomepageFooter from './components/homepage/HomepageFooter';

function Homepage(){
    return(
        <div>
            <HomepageHeader />
            <Presentation />
            <News />
            <HomepageFooter />
        </div>
    )
}
```

Definir los estilos CSS



Los estilos de las páginas fueron agrupados en la carpeta de estilos, se hizo uso de medidas responsivas medidas responsivas, contenedores como flex y grid, y variables de color para una mejor administracion de los estilos en caso de un cambio de diseño.

Tipos de dato utilizados para el proyecto



Las interfaces en typescript son usadas para definir las características de una entidad u objetos, cabe mencionar que estas desaparecen cuando se inicia con el proceso de construcción del aplicativo al transpilar el código de typescript a javascript.

Interfaz para Post

```
export interface Posts{
  id_post?: string;
  titulo: string;
  fecha_publicacion: string;
  contenido: string;
  nickname_usuario: string
}
```

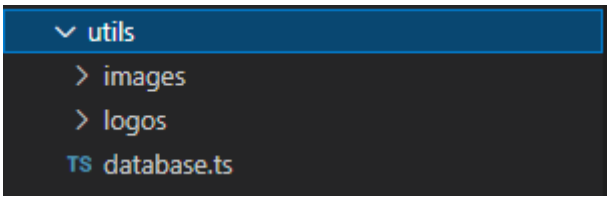
Interfaz para User

```
export interface User {
    nickname: string;
    email: string;
}
```

Interfaz usuarios

```
export interface usuarios{
    id_usuarios?:string;
    nickname:string;
    email?:string;
    password:string;
    login?:string;
}
```

Configurar conexion a BASE DE DATOS postgresql

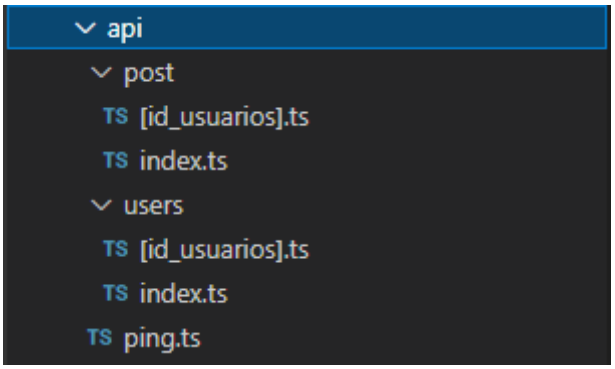


La conexión a la base de datos se sitúa en la carpeta utils, de esta manera podemos hacer uso de ella cada vez que se requiera. Dado que para la conexión no se necesitan componentes, la extensión de “database” será ts por TypeScript.

```
import {Pool} from 'pg'

let conn:any

if(!conn){
    conn = new Pool({
        user: 'postgres',
        password: 'toor',
        host: 'localhost',
        port: 5432,
        database:'Blog'
    });
}

export {conn}
```

El módulo utilizado para conectarse con la base de datos postgres es ‘pg’. Para este caso se importa utilizando “Pool” de tal forma que tengamos un conjunto de conexiones que podemos utilizar luego.

Crear Api rest



De acuerdo a la estructura planteada, en la carpeta API estarán contenidos los recursos necesarios para el backend.

Peticiones asíncronas para interactuar frontend y backend

```
import { NextApiRequest, NextApiResponse } from "next";
import {conn} from 'src/utils/database'

export default async (req:NextApiRequest,res:NextApiResponse) => {
}
```

Importamos desde NextJs tanto NextApiRequest como NextApiResponse, estos parámetros son interfaces que permiten detallar los objetos request y response.

Las peticiones especificadas a continuación, nos permitirán detallar qué método requerimos para realizar determinada tarea desde el backend, de tal forma que podamos acceder a través de una URL.

Peticiones para Posts

[id.usuarios]

```
const {method,query} = req
    switch (method) {
        case "GET":
            try {
                const text = 'SELECT titulo, fecha_publicacion, contenido, estatus FROM post as ps where
ps.id_usuarios = $1'
                const values = [query.id_usuarios]
                const result = await conn.query(text,values)
                console.log(query.id_usuarios)
                if (result.rows.length ===0)
                {
                    return res.status(404).json({message:"Post no found"})
                }
                return res.status(200).json(result.rows)
            } catch (error:any) {
                return res.status(500).json({message:error.message})
            }
        case "PUT":
            res.status(200).json("update data")
            break;
        case "DELETE":
            res.status(200).json("delete data")
            break;
        default:
            res.status(400).json("Invalid method")
            break;
    }
```

index

```
const {method,body} = req
    switch (method) {
        case "GET":
            try {
                const query = "SELECT * FROM post";
                const  response = await conn.query(query);
                return res.status(200).json(response.rows)
            } catch (error:any) {
                return  res.status(500).json({error: error.message});
            }
            case "POST":
            try {
                const {nickname,titulo,contenido,estatus} = body;
                const query = 'INSERT INTO post (id_usuarios,titulo,contenido,estatus) VALUES ((SELECT
id_usuarios FROM usuarios WHERE nickname=$1),$2,$3,$4) RETURNING *'
                const values =[nickname,titulo,contenido,estatus]
                const respose = await conn.query(query,values)
                return res.status(200).json(respose.rows)
            } catch (error:any) {
                return  res.status(500).json({error: error.message});
            }
        default:
            res.status(400).json("Invalid method")
            break;
    }
```

Peticiones para Usuarios

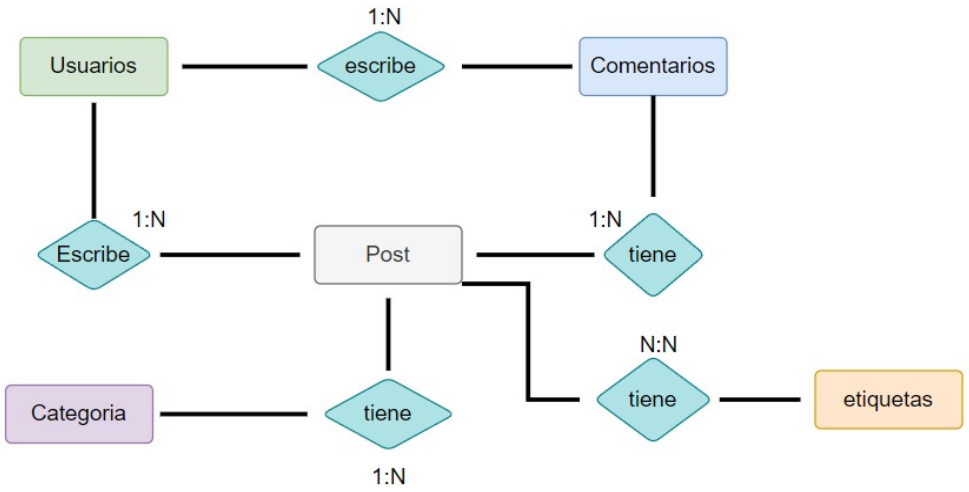
[id.usuarios]

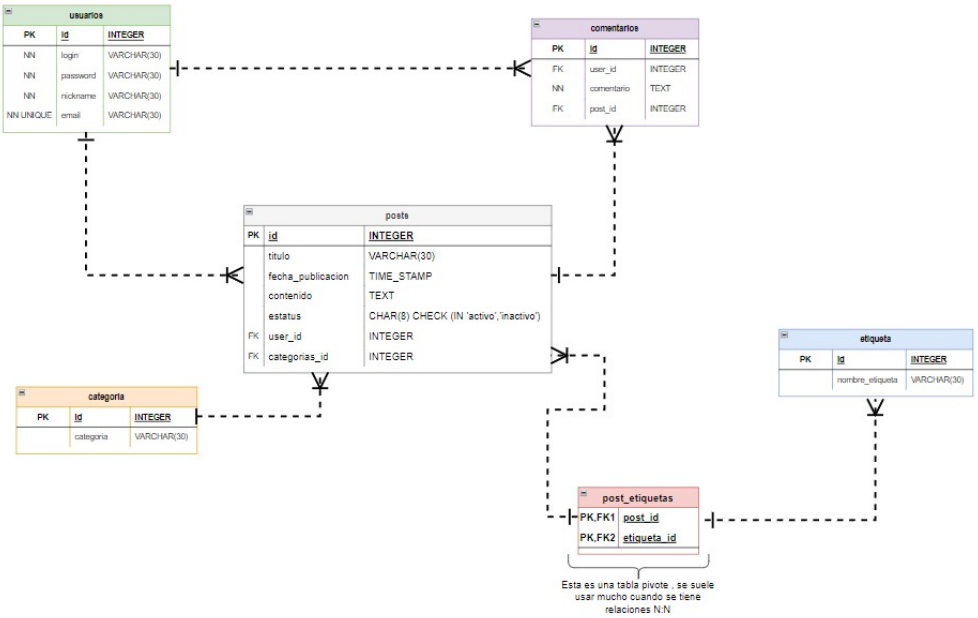
```
    const {method,query} = req
    switch (method) {
        case "GET":
            try {
                const text = 'SELECT * FROM usuarios WHERE id_usuarios = $1'
                const values = [query.id_usuarios]
                const result = await conn.query(text,values)
                console.log(result)
                if (result.rows.length ===0)
                {
                    return res.status(404).json({message:"User no found"})
                }
                return res.status(200).json(result.rows[0])
            } catch (error:any) {
                return res.status(500).json({message:error.message})
            }
        case "PUT":
            res.status(200).json("update data")
            break;
        case "DELETE":
            res.status(200).json("delete data")
            break;
        default:
            res.status(400).json("Invalid method")
            break;
    }
```

index

```
const {method,body} = req
    switch (method) {
        case "GET":
            try {
                const query = "SELECT * FROM USUARIOS";
                const  response = await conn.query(query);
                return res.status(200).json(response.rows)
            } catch (error:any) {
                return  res.status(500).json({error: error.message});
            }
        case "POST":
            try {
                const {nickname,email,password} = body;
                if (!email)
                {
                    const query = 'SELECT * FROM usuarios WHERE nickname = $1 AND password = MD5($2)'
                    const values =[nickname,password]
                    const respose = await conn.query(query,values)
                    console.log(respose.rows[0])
                    if(respose.rows.length == 0){
                        return res.status(400).json({error:"Usuario no encontrado"})
                    }
                    return res.status(200).json(respose.rows[0])
                }
                else{
                    const query = 'INSERT INTO usuarios (nickname,email,password) VALUES ($1,$2,MD5($3))
RETURNING *'
                    const values =[nickname,email,password]
                    const respose = await conn.query(query,values)
                    return res.status(200).json(respose.rows)
                }
            } catch (error:any) {
                return  res.status(500).json({error: error.message});
            }
        default:
            res.status(400).json("Invalid method")
            break;
    }
```

Base de datos





3. Conclusiones

- Next.js es un framework que funciona con la ideología de react, tanto su componentes como otros módulos que ayudan en el proceso de desarrollo.
- Next.js incluye algunas etiquetas y módulos para obtener un mejor rendimiento en el aplicativo, además de renderización del lado de servidor, enrutamiento dinámico, entre otras funcionalidades que aumentan el rendimiento de las páginas web.
- Los componentes son el principal enfoque tanto de React como de Next, ayudan a tener una mejor organización del proyecto, a reutilizar los componentes, y a construir páginas web de una forma más rápida y organizada.

4. Observaciones

- El framework Next.js tiene una gran cantidad de ventajas, pero también sus desventajas como el no tener un administrador de estados de componentes integrado, por lo que se debe hacer uso de otras herramientas externas, como Redux.js.
- Los componentes de react pueden ser desarrollados con lenguaje de JavaScript o Typescript, pero se debe tomar en cuenta que ciertas etiquetas y funciones cambian o se diferencian al usar un lenguaje del otro.