



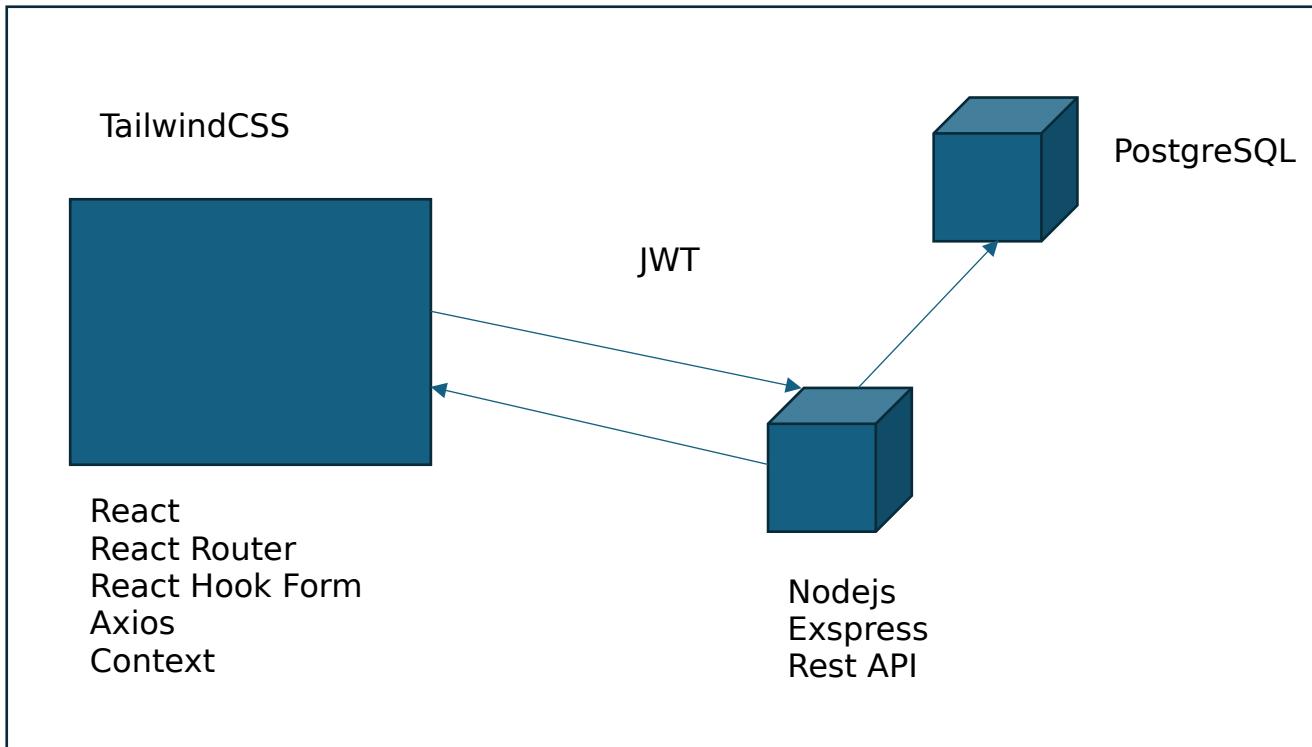
React Postgre SQL y Node Js con autenticación y CRUD

Desarrollo de Software I (Universidad del Valle Colombia)



Scan to open on Studocu

React PostgreSQL y NodeJs con autenticación y CRUD



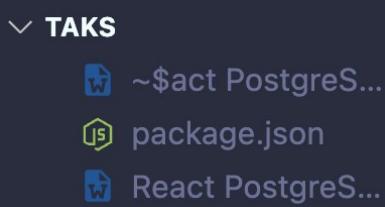
Backend Setup

Lo primero es crear el contenedor del proyecto y lo colocamos en nuestro editor de código. Después de eso abrimos el terminal del VsCode y ejecutamos el código que se ve a continuación.

```
● danielquintero@MacBook-Air-de-Daniel Taks % npm init -y
Wrote to /Users/danielquintero/Desktop/Proyectos 2024/Vite/Taks/package.json:

{
  "name": "taks",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Se crea un archivo package.json



Instalemos las librerías para la comunicación del Back con el Front

```
● danielquintero@MacBook-Air-de-Daniel Taks % sudo npm i express cookie-parser cors morgan pg zod md5 jsonwebtoken express-promise-router
```

Necesitamos los módulos de dependencias de desarrollo

```
● danielquintero@MacBook-Air-de-Daniel Taks % sudo npm i nodemon eslint -D
```

Vamos a configurar un comando de ejecución y un type module para los import y export del proyecto, creamos una carpeta src y dentro creamos un index.js y solo mandamos un mensaje por el momento.

The terminal shows the configuration of the 'package.json' file and the creation of a 'src' directory with an 'index.js' file.

```
package.json > {} scripts > start
1  {
2    "name": "taks",
3    "version": "1.0.0",
4    "main": "src/index.js",
5    "type": "module",
6    "scripts": {
7      "dev": "nodemon .",
8      "start": "node ."
9    },
10   }

< TAKS >
  > node_modules
  < src >
    index.js
```

```
src > index.js
1  console.log('Hola desde el back')
```

Cada vez que hagamos un cambio en el index.js el nodemon lo detecta y lo muestra

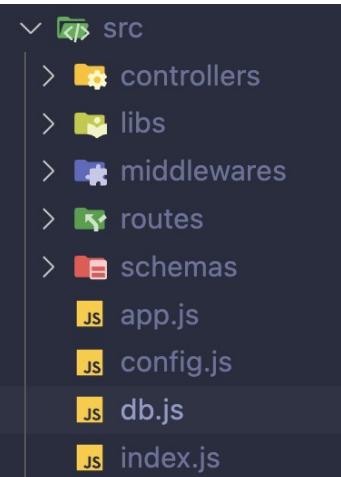
```

○ danielquintero@MacBook-Air-de-Daniel Taks % npm run dev
> taks@1.0.0 dev
> nodemon .

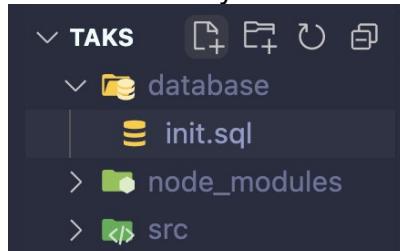
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node .
Hola desde el back
[nodemon] clean exit - waiting for changes before restart

```

En el src vamos a crear una estructura de carpetas como se ve a continuación y otros archivos dentro de src



Aparte de esto vamos a crear una carpeta llamada database, hace referencia a la base de datos y también le creamos un archivo llamado init.sql



Vamos a ir haciendo nuestro repositorio entonces detenemos el run dev y ejecutamos el código que vemos a continuación y de una vez por fuera de todas las carpetas vamos a crear el .gitignore para que no tenga en cuenta la carpeta node_modules para cuando lo vayamos a subir a algún repositorio o contenedor

```

● danielquintero@MacBook-Air-de-Daniel Taks % git init
Initialized empty Git repository in /Users/danielquintero/Desktop/Proyectos 2024/Vite/Taks/.git/

```

File .gitignore:

```

database
node_modules
src
.gitignore

```

Hagamos los siguientes comandos de git

- danielquintero@MacBook-Air-de-Daniel Taks % git add .
- danielquintero@MacBook-Air-de-Daniel Taks % git status
- danielquintero@MacBook-Air-de-Daniel Taks % git commit -m "first commit"

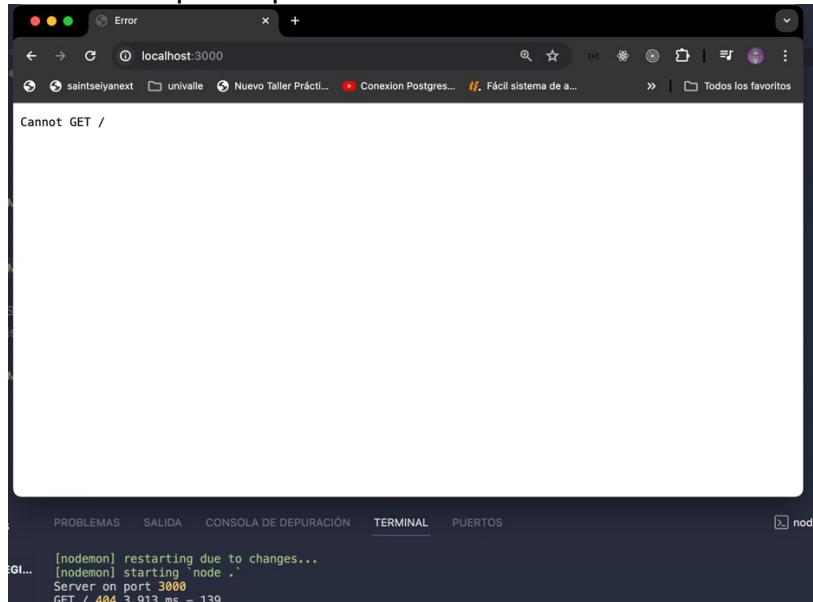
Express Server

Vamos a mostrar si desde el front se están haciendo peticiones a lo cual vamos a utilizar el paquete de Morgan entonces en el app.js vamos a configurar e importar las librerías instaladas anteriormente

```
src > [JS] app.js > [✖] default
1   import express from "express";
2   import morgan from "morgan";
3
4   const app = express()
5   app.use(morgan('dev'))
6
7
8   export default app

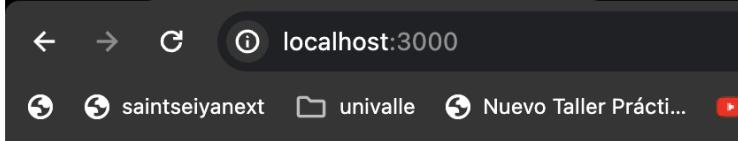
src > [JS] index.js
1   import app from './app.js'
2
3   app.listen(3000)
4
5   console.log(['Server on port', 3000])
```

Ahora en el navegador colocamos la url localhost:3000 y recuerde que debemos ejecutar el comando npm run dev para que veamos las peticiones desde el front hacia el back, recuerde que sale un error porque no tenemos rutas configuradas y el no sabe que respuesta dar.



Cuando volvemos a hacer una petición desde el navegador obtenemos una respuesta que nosotros hemos configurado en el app.js

```
src > [js] app.js > [o] default
1 import express from "express";
2 import morgan from "morgan";
3
4 const app = express()
5 app.use(morgan('dev'))
6 app.get('/', (req, res) => res.json({message: "Welcome to my API"}))
7
8 export default app
```



Vamos a dejar nuestro app.js de la siguiente manera

```
src > [js] app.js > ...
1 import express from "express";
2 import morgan from "morgan";
3
4 const app = express()
5 app.use(morgan('dev'))
6 app.use(express.json())
7 app.use(express.urlencoded({ extended: false }))
8 app.get('/', (req, res) => res.json({message: "Welcome to my API"}))
9
10 app.use((err, req, res, next) => {
11   res.status(500).json({
12     status: "error",
13     message: err.message
14   })
15 })
16
17 export default app
```

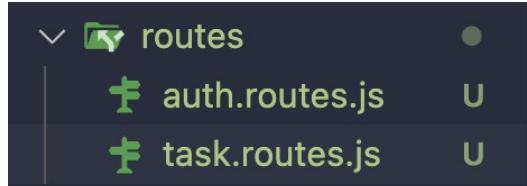
Y vamos a hacer los cambios se guarden en el git

```
● danielquintero@MacBook-Air-de-Daniel Taks % git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   "React PostgreSQL y NodeJs con autenticaci\303\263n y CRUD.docx"
      modified:   src/app.js
      modified:   src/index.js

no changes added to commit (use "git add" and/or "git commit -a")
● danielquintero@MacBook-Air-de-Daniel Taks % git add .
● danielquintero@MacBook-Air-de-Daniel Taks % git commit "express basic setup"
  error: pathspec 'express basic setup' did not match any file(s) known to git
● danielquintero@MacBook-Air-de-Daniel Taks % git commit -m "express basic setup"
```

BackEnd Routes

Creamos las rutas, recuerde que creamos una carpeta para el manejo de estas y entonces vamos a crear dos archivos uno para las rutas del crud y otras para las de autenticación.



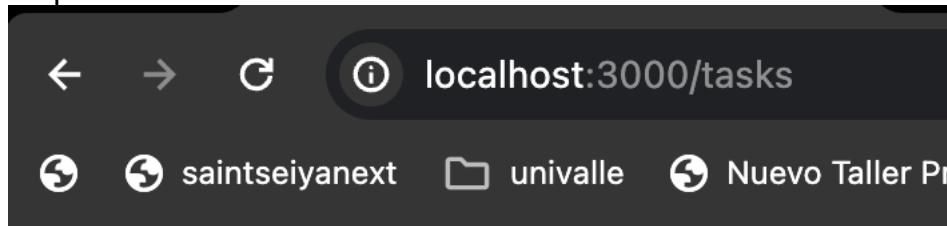
Configuramos nuestra ruta para las tareas

```
src > routes > task.routes.js > ...
1 import { Router } from "express";
2
3 const router = Router()
4
5 router.get('/tasks', (req, res) => res.send('Obteniendo tareas'))
6
7
8 export default router
```

Luego lo tenemos que usar en la aplicación principal app.js

```
src > app.js > ...
1 import express from "express";
2 import morgan from "morgan";
3 import taskRoutes from './routes/task.routes.js'
4
5 const app = express()
6 app.use(morgan('dev'))
7 app.use(express.json())
8 app.use(express.urlencoded({ extended: false }))
9
10 app.get('/', (req, res) => res.json({message: "Welcome to my API"}))
11
12 app.use(taskRoutes)
13
14 app.use((err,req,res,next) => {
15   res.status(500).json({
16     status: "error",
17     message: err.message
18   })
19 })
20
21 export default app
```

Recuerde que desde el navegador si damos la ruta ya debemos obtener una respuesta



Obteniendo tareas

Creemos las diferentes funcionalidades que va tener nuestra aplicación

```
src > routes > task.routes.js > ...
1 import { Router } from "express";
2
3 const router = Router()
4
5 router.get('/tasks', (req, res) => res.send('Obteniendo tareas'))
6
7 router.get('/tasks/:id', (req, res) => res.send('Obteniendo tarea unica'))
8
9 router.post('/tasks', (req, res) => res.send('Creando tarea'))
10
11 router.put('/tasks/:id', (req, res) => res.send('Actualizando tarea unica'))
12
13 router.delete('/tasks/:id', (req, res) => res.send('Eliminando tarea unica'))
14
15
16 export default router
```

Configuración de rutas de autenticación

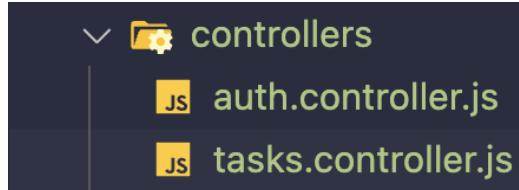
```
src > routes > auth.routes.js > default
1 import { Router } from "express"
2
3 const router = Router()
4
5 router.post('/singin',(req,res)=>res.send('Ingresando'))
6
7 router.post('/signup',(req,res)=>res.send('Registrando '))
8
9 router.post('/singout',(req,res)=>res.send('Cerrando sesion'))
10
11 router.get('/profile',(req,res)=>res.send('Perfil del Usuario'))
12
13 export default router
```

Recuerde que cada vez que hace un cambio la aplicación principal lo debe saber

```
src > app.js > default
1 import express from "express";
2 import morgan from "morgan";
3 import taskRoutes from './routes/task.routes.js'
4 import authRoutes from './routes/auth.routes.js'
5
6 const app = express()
7 //middleware
8 app.use(morgan('dev'))
9 app.use(express.json())
10 app.use(express.urlencoded({ extended: false }))
11 //Routes
12 app.get('/', (req, res) => res.json({message: "Welcome to my API"}))
13 app.use('/api',authRoutes)
14 app.use('/api',taskRoutes)
15 //Error Handler
16 app.use((err,req,res,next) => {
17   res.status(500).json({
18     status: "error",
19     message: err.message
20   })
21 })
22
23 export default app|
```

Controllers

Separar las funciones cuando llegan de alguna ruta, dentro de la carpeta controllers vamos a crear dos archivos



Vamos a task.routes.js y vamos a cortar la función (req,res) y la vamos a pegar dentro de nuestro tasks.controller.js el controller queda de la siguiente manera

```
src > controllers > JS tasks.controller.js > [e] getAllTasks
1
2 export const getAllTasks = (req, res) => res.send('Obteniendo tareas')
3
4 export const getTask = (req, res) => res.send('Obteniendo tarea unica')
5
6 export const createTask = (req, res) => res.send('Creando tarea')
7
8 export const updateTask = (req, res) => res.send('Actualizando tarea unica')
9
10 export const deleteTask = (req, res) => res.send('Eliminando tarea unica')
```

Y las rutas nuevas quedan asi

```
src > routes > TS task.routes.js > ...
1 import { Router } from "express";
2 import { createTask, deleteTask, getAllTasks, getTask, updateTask } from '../controllers/tasks.controller.js'
3
4 const router = Router()
5
6 router.get('/tasks', getAllTasks)
7
8 router.get('/tasks/:id', getTask)
9
10 router.post('/tasks', createTask)
11
12 router.put('/tasks/:id', updateTask)
13
14 router.delete('/tasks/:id', deleteTask)
15
16 export default router
```

Ahora hacemos lo mismo para nuestras rutas de autenticación

```

src > controllers > auth.controller.js > [o] profile
1   export const singin = (req,res)=>res.send('Ingresando')
2
3   export const singup = (req,res)=>res.send('Registrando ')
4
5   export const singout = (req,res)=>res.send('Cerrando sesion')
6
7   export const profile = (req,res)=>res.send(['Perfil del Usuario'])

src > routes > auth.routes.js > [o] default
1   import { Router } from 'express'
2   import { profile, singin, singout, singup } from '../controllers/auth.controller.js'
3
4   const router = Router()
5
6   router.post('/singin', singin)
7
8   router.post('/singup', singup)
9
10  router.post('/singout', singout)
11
12  router.get('/profile', profile)
13
14  export default router

```

Bases de datos crear Tarea

Como conectar el proyecto a las BD postgres, vamos a hacer la conexión desde el archivo db.js

```

src > db.js > [o] pool
1   import pg from 'pg'
2
3   export const pool = new pg.Pool([
4     port: 5432,
5     host: 'localhost',
6     user: 'postgres',
7     password: 'postgres'
8   ])
9
10  pool.on('connect', ()=>{
11    console.log('Database connected')
12  })

```

Ahora en el index.js que es de donde estamos recibiendo respuesta del server vamos a hacer que nos muestre por el momento la fecha para verificar la conexión con postgres, pero recuerde que lo debe tener instalado primero

```
src > JS index.js > ...
1 import app from './app.js'
2 import {pool} from './db.js'
3
4 pool.query('select now()', (err, res) =>{
5   console.log(err, res)
6   app.listen(3000)
7   console.log('Server on port', 3000)
8   pool.end()
9 })
```

Y pues como solo quiero ver la fecha yo veo que la respuesta se guarda en rows

```
src > JS index.js > ...
1 import app from './app.js'
2 import {pool} from './db.js'
3
4 pool.query('select now()', (err, res) =>{
5   console.log(err, res.rows)
6   app.listen(3000)
7   console.log('Server on port', 3000)
8   pool.end()
9 })
```

Pero no es necesario tener la conexión al inicio esto es solo para verificar la conexión

```
src > JS index.js
1 import app from './app.js'
2
3
4 app.listen(3000)
5 console.log('Server on port', 3000)
```

Vamos a crear la tabla, para esto lo hacemos en el archivo init.sql

```
database > SQL init.sql
1 create table task(
2   id serial primary key,
3   title varchar(255) unique not null,
4   description text
5 );
```

Vamos a instalar un programa administrador de motores de bases de datos llamado DBeaver <https://dbeaver.io/download/> y los descarga para su sistema operativo.

Después de instalado se nos abre una ventana y damos clic en donde hay un conector con un símbolo de +



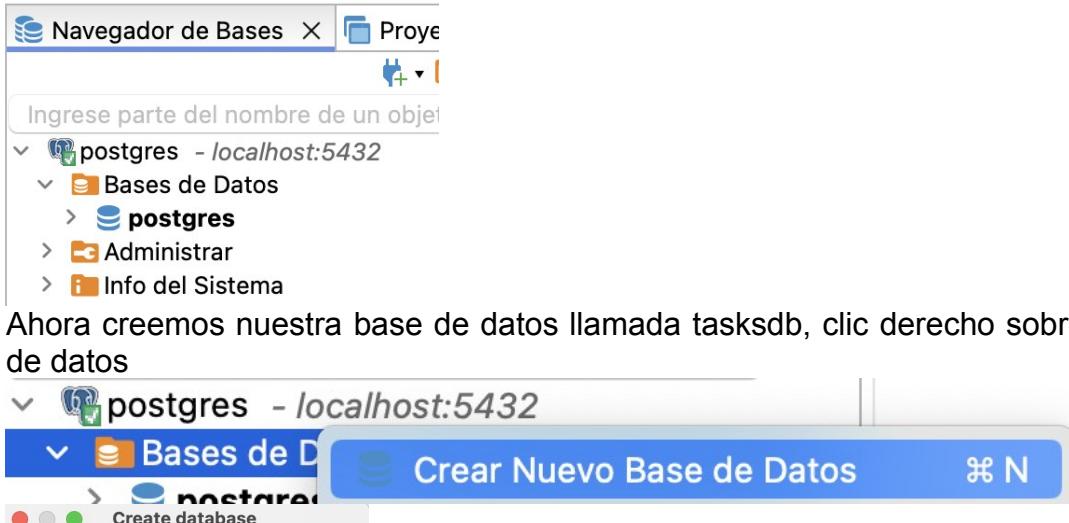
Se nos despliega una nueva ventana y seleccionamos postgres, recuerde que ya debe estar instalado

Seleccione su base de datos

Crear nueva conexión a base de datos. E



Y finalizar y te sale que postgres esta conectado



Ahora creamos nuestra base de datos llamada tasksdb, clic derecho sobre bases de datos



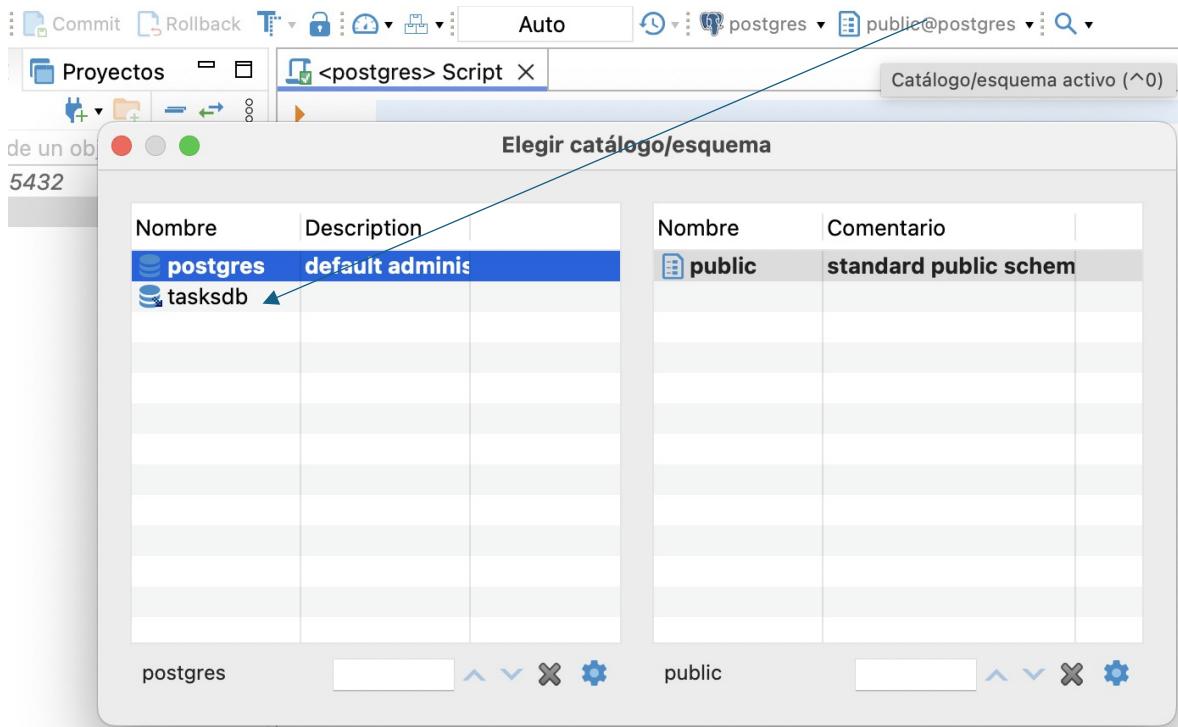
Aceptar



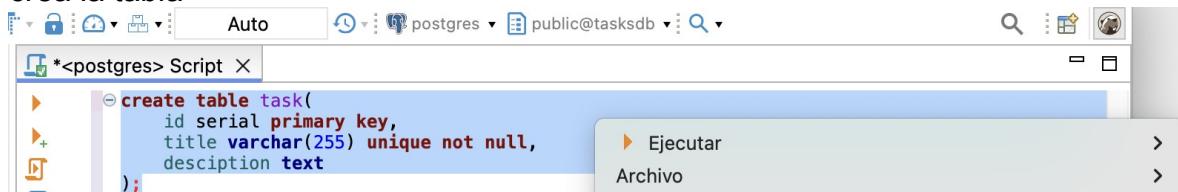
Ya esta creada nuestra base de datos, crearemos la tabla, en el programa vamos a dar clic en sql



Damos clic en donde dice public@postres y nos sale que base de datos vamos a trabajar



Elegimos tasksdb, para poder ejecutar el script de la tabla solo debemos seleccionar el código sql y le damos clic derecho ejecutar script y con eso se nos crea la tabla



Y la tabla queda creada



Antes de continuar voy a hacer un cambio en las rutas en el app.js

```
src > js app.js > ...
16  app.use('/api', authRoutes)
17  app.use('/api', taskRoutes)
```

Y en el archivo db.js debemos colocar la base de datos a donde vamos a enviar los datos

```

src > db.js > ...
1 import pg from 'pg'
2 export const pool = new pg.Pool({
3   port: 5432,
4   host: 'localhost',
5   user: 'postgres',
6   password: 'postgres',
7   database: 'tasksdb'
8 })
9 pool.on('connect', ()=>{
10   console.log('Database connected')
11 })

```

Mi archivo db.js pedirá siempre la conexión

```

src > db.js > ...
1 import pg from 'pg'
2 export const pool = new pg.Pool({
3   port: 5432,
4   host: 'localhost',
5   user: 'postgres',
6   password: 'postgres',
7   database: 'tasksdb'
8 })
9 pool.on('connect', ()=>{
10   console.log('Database connected')
11 })

```

Vamos a insertar nuestro primer registro en la base de datos, recuerde que para esto es el controlador tasks.controller.js

```

src > controllers > tasks.controller.js > [o] createTask
1 import {pool} from '../db.js'
2
3 export const getAllTasks = (req, res) => res.send('Obteniendo tareas')
4
5 export const getTask = (req, res) => res.send('Obteniendo tarea unica')
6
7 export const createTask = async (req, res) => {
8   const { title, description } = req.body
9   console.log( title, description )
10  //db insert
11  const {rows} = await pool.query(['insert into task (title, description) values ($1, $2)', [
12    [title, description]
13  ]])
14  console.log(rows)
15  res.send('creando tarea')
16}
17
18 export const updateTask = (req, res) => res.send('Actualizando tarea unica')
19
20 export const deleteTask = (req, res) => res.send('Eliminando tarea unica')

```

Se está mandando en el controlador \$1, \$2 ya que nosotros estamos enviando la información desde el propio backend, para esto instale la extensión de postman en el vscode

POST | http://localhost:3000/api/tasks

Params | Authorization | Headers (9) | **Body** | Pre-request Script | Tests | Settings | Code Cookies | Beautify

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```

1 ...
2 ... "title": "mi tarea 1",
3 ... "description": "creando mi primer tarea"
4

```

Body | Cookies | Headers (7) | Test Results | Status: 200 OK Time: 55 ms Size: 240 B | HTML | Preview |

Pretty | Raw | Preview | HTML |

1 creando tarea

Miramos en la bd y ya encontramos el registro que guardamos

`select * from task`

c1 X		
Select * from task Enter a SQL expression to filter results		
1 id	2 title	3 description
4	mi tarea 1	creando mi primer ta

Con el insert anterior vemos que cuando se nos genera un error que puede ser porque mandamos un campo en blanco o volvemos a mandar el mismo dato y recuerde que le decimos que el campo title no se puede repetir y el server se cae y para que esto no suceda debemos crear un try catch

```

src > controllers > tasks.controller.js > [o] createTask
7 export const createTask = async (req, res) => {
8   const { title, description } = req.body
9   try {
10     const {rows} = await pool.query('insert into task (title, description) values ($1, $2)', [
11       title, description
12     ])
13     console.log(rows)
14     res.send('creando tarea')
15   } catch (error) {
16     console.log(error)
17     return res.send('Algo salio mal')
18   }
19 }

```

Pero este código puede mejorar mucho mas para manejo de errores básicos y mostrar solo lo insertado

```
src > controllers > tasks.controller.js > [e] createTask
  7  export const createTask = async (req, res, next) => {
  8    const { title, description } = req.body
  9    try {
10      const result = await pool.query('insert into task (title, description) values ($1, $2) returning *',
11                                      [title, description])
12    }
13    res.json(result.rows[0])
14  } catch (error) {
15    console.log(error)
16    if(error.code === "23505"){
17      return res.send('La tarea ya existe')
18    }
19    next(error)
20  }
21 }
```

Obtener tareas

Como obtener las tareas creadas, pero también vamos a hacer un cambio en el task.routes.js donde vamos a usar el express-promise-router, para el manejo de los errores

```
src > routes > task.routes.js > ...
  1  import Router from "express-promise-router";
  2  import { createTask, deleteTask, getAllTasks, getTask, updateTask } from '../controllers/tasks.controller.js'
```

```
src > controllers > tasks.controller.js > result
3  export const getAllTasks = async (req, res, next) => {
4    const result = await pool.query('select * from task')
5    return res.json(result.rows)
6 }
```

Vamos a mandar el error del insert con un json y un status que son los errores de server

```
src > controllers > tasks.controller.js > createTask > message
10 export const createTask = async (req, res, next) => {
11   const { title, description } = req.body
12   try {
13     const result = await pool.query('insert into task (title, description) values ($1, $2) returning *',
14       [title, description]
15     )
16     res.json(result.rows[0])
17   } catch (error) {
18     console.log(error)
19     if(error.code === "23505"){
20       return res.status(409).json({
21         message: 'Ya existe una tarea con este titulo'
22       })
23     }
24   }
25 }
26 }
```

Obtener una tarea

Consulta de una sola tarea

```
src > controllers > tasks.controller.js > [o] getTask > ⚡ message
  8  export const getTask = async (req, res) => {
  9    const result = await pool.query("select * from task where id = $1", [
10      req.params.id,
11    ])
12    if(result.rowCount === 0){
13      return res.status(404).json({
14        message: "No existe una tarea con ese id"
15      })
16    }
17    return res.json(result.rows[0])
18 }
```

Eliminar Tarea

```
src > controllers > tasks.controller.js > [o] deleteTask
  40  export const deleteTask = async (req, res) => {
  41    const result = await pool.query('delete from task where id = $1', [req.params.id])
  42    console.log(result)
  43    if(result.rowCount === 0){
  44      return res.status(404).json({
  45        message: "La tarea que estas eliminando ya fue eliminada o no existe"
  46      })
  47    }
  48    return res.sendStatus(204)
  49 }
```

Actualizar Tarea

```
src > controllers > tasks.controller.js > [o] updateTask
  38  export const updateTask = async(req, res) => {
  39    const id = req.params.id
  40    const { title, description } = req.body
  41    const result = await pool.query(
  42      'update task set title = $1, description = $2 where id = $3 returning *',
  43      [title, description, id]
  44    )
  45    if(result.rowCount === 0){
  46      return res.status(404).json({
  47        message: 'No existe una tarea con ese id'
  48      })
  49    }
  50    return res.json(result.rows[0])
  51 }
```

Registro de usuarios

Para registrar los usuarios vamos a crear una tabla para el manejo de los usuarios en nuestro archivo init.sql

```

database > init.sql
 7  create table users(
 8    id serial primary key,
 9    name varchar(255) not null,
10    password varchar(255) not null,
11    email varchar(255) unique not null,
12    created_at timestamp default CURRENT_TIMESTAMP,
13    updated_at timestamp default CURRENT_TIMESTAMP,
14 )

```

Recuerde que esto se ejecuta ya sea en un administrador de bases de datos o en el mismo motor esto ya es a su gusto. Recuerde que nosotros colocamos el correo como unique entonces hacemos el cambio de express en el auth.route.ts

```

src > routes > auth.routes.js > ...
 1 import Router from 'express-promise-router'
 2 import { profile, singin, singout, singup } from '../controllers/auth.controller.js'
 3
 4 const router = Router()
 5
 6 router.post('/singin', singin)
 7
 8 router.post('/singup', singup)
 9
10 router.post('/singout', singout)
11
12 router.get('/profile', profile)
13
14 export default router

```

Entonces guardemos el primer usuario en nuestra base de datos

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `http://localhost:3000/api/singup`. Below the header, there is a method selector set to `POST`. The main area shows the request details:

- Params**: none
- Authorization**: None
- Headers**: (9)
- Body** (highlighted): raw
- Pre-request Script**: None
- Tests**: None
- Settings**: None

The `Body` section contains the following JSON payload:

```

1 {
2   "name": "daniel",
3   "email": "djuanrique@gmail.com",
4   "password": "1234567"
5 }

```

Si intentamos guardar el mismo usuario recuerde que para eso tenemos la librería `express-promise-router` y cuando enviamos el mismo usuario el servidor ya no se cae y nos genera el error

The screenshot shows a Postman interface with the following details:

- HTTP Method:** POST
- URL:** http://localhost:3000/api/signup
- Body:** raw JSON (selected)
- Request Body:**

```
1 {  
2   "name": "daniel",  
3   "email": "djuanrique@gmail.com",  
4   "password": "1234567"  
5 }
```

- Status:** 500 Internal Server Error
- Response Body (Pretty JSON):**

```
1 {}  
2 "status": "error",  
3 "message": "duplicate key value violates unique constraint \\"users_email_key\\"
```

Text below the screenshot:

Cuando estamos creando los usuarios podemos observar la contraseña creada y no esta encriptada para eso vamos a instalar una librería

○ danielquintero@MacBook-Air-de-Daniel Taks % sudo npm i bcrypt

Importamos la librería que acabamos de instalar

```
src > controllers > auth.controller.js > singup
1 import bcrypt from 'bcrypt'
2 import {pool} from '../db.js'
3
4 export const singin = (req,res)=>res.send('Ingresando')
5
6 export const singup = async (req,res)=>{
7     const {name, email, password} = req.body
8     try {
9         const hashedPassword = await bcrypt.hash(password, 10)
10        console.log(hashedPassword)
11        const result = await pool.query
12        (
13            'insert into users(name, email, password) values($1,$2,$3)', 
14            [name, email, hashedPassword]
15        );
16        console.log(result)
17        return res.json(result.rows[0])
18    } catch (error) {
19        if(error.code === '23505'){
20            return res.status(400).json({
21                message: 'El email ya esta registrado'
22            })
23        }
24    }
25 }
```

JSON WEB TOKEN

Crear un token para el front end es para decir que el usuario ya se registro, en la carpeta libs creamos un archivo llamado jwt.js

```

src > libs > js jwt.js > ...
1 import jwt from "jsonwebtoken";
2
3 export const createAccessToken = (payload) => {
4   return new Promise((resolve, reject) => {
5     jwt.sign(
6       payload,
7       "xyz123",
8       {
9         expiresIn: "1d",
10      },
11      (err, token) => {
12        if (err) reject(err);
13        resolve(token);
14      }
15    );
16  });
17};

src > controllers > js auth.controller.js > [o] singup
3 | import { createAccessToken } from '../libs/jwt.js'
4 |
5 export const singin = (req,res)=>res.send('Ingresando')
6
7 export const singup = async (req,res)=>{
8   const {name, email, password} = req.body
9   try [
10     const hashedPassword = await bcrypt.hash(password, 10)
11     console.log(hashedPassword)
12     const result = await pool.query
13     (
14       'insert into users(name, email, password) values($1,$2,$3)',
15       [name, email, hashedPassword]
16     );
17
18     const token = await createAccessToken({ id: result.rows[0].id })
19     console.log(result)
20     return res.json(result.rows[0])
21     //return res.json({"token": token})
22   } catch (error) {
23     if(error.code === '23505'){
24       return res.status(400).json({
25         message: 'El email ya esta registrado'
26       })
27     }
28   }
29 }

```

Cookies

Guardar el token en el navegador, esto es para que el frontend no lo tenga que solicitar, antes de continuar instalaremos la librería siguiente

```
● danielquintero@MacBook-Air-de-Daniel Taks % sudo npm i --save-dev @types/jsonwebtoken
Password:
```

Quiero mostrar los headers de todos los getAllTasks para ver a que cookie esta asociado

```
src > controllers > tasks.controller.js > [o] getAllTasks
<
3  export const getAllTasks = async (req, res, next) => {
4    console.log(req.headers)
5    const result = await pool.query('select * from task')
6    return res.json(result.rows)
7 }
```

```
src > controllers > auth.controller.js > [o] singout
6
7  export const singup = async (req,res)=>{
8    const {name, email, password} = req.body
9    try {
10      const hashedPassword = await bcrypt.hash(password, 10)
11      console.log(hashedPassword)
12      const result = await pool.query(
13        (
14          'insert into users(name, email, password) values($1,$2,$3) Returning *',
15          [name, email, hashedPassword]
16        );
17
18      const token = await createAccessToken({ id: result.rows[0].id })
19      res.cookie("token", token, {
20        httpOnly: true,
21        sameSite: 'none',
22        maxAge: 24 * 60 * 60 * 1000 //1 dia
23      })
24      return res.json(result.rows[0])
25      //return res.json({"token": token})
26    } catch (error) {
27      if(error.code === '23505'){
28        return res.status(400).json({
29          message: 'El email ya esta registrado'
30        })
31      }
32    }
33 }
```

Ya en nuestro auth controller vamos a decir cuanto dura el token

Vamos a crear un archivo en la carpeta middlewares un auth.middleware.js

```

src > middlewares > auth.middleware.js > [o] isAuth
1
2   export const isAuth = (req, res, next)=>{
3     console.log(req.headers)
4     next()
5   }

```

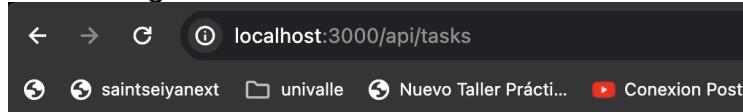
Utilicemos la librería cookie-parser en nuestro app.js

```

src > app.js > ...
3   import cookieParser from 'cookie-parser'
4
5   import taskRoutes from './routes/task.routes.js'
6   import authRoutes from './routes/auth.routes.js'
7
8   const app = express()
9   //middleware
10  app.use(morgan('dev'))
11  app.use(cookieParser())

```

Hagamos que la cookie sea obligatoria para poder consultar nuestras tareas del usuario logueado



```

1  // 20240520161239
2  // http://localhost:3000/api/tasks
3
4  {
5    "message": "No estas autorizado"
6  }

```

En el middleware podemos generar este error

```

src > middlewares > auth.middleware.js > [o] isAuth > ✎ message
1
2   export const isAuth = (req, res, next)=>{
3     const token = req.cookies.token
4     if(!token){
5       return res.status(401).json([
6         message: 'No estas autorizado'
7       ])
8     }
9     next()
10 }

```

Pero el middleware debe devolver el token o sea hacer el proceso contrario, donde muestro que usuario hizo la petición.

```
src > middlewares > auth.middleware.js > ...
1 import jwt from 'jsonwebtoken'
2
3 export const isAuth = (req, res, next) => {
4     const token = req.cookies.token
5     if(!token){
6         return res.status(401).json({
7             message: 'No estas autorizado'
8         })
9     }
10
11     jwt.verify(token, 'xyz123', (err, decoded) =>{
12         if(err) return res.status(401).json({
13             message: 'No estas autorizado'
14         })
15         req.userId = decoded.id
16         next()
17     })
18
19 }
```

Entonces cuando hacemos la solicitud de la función getAllTasks nos sale quien es el usuario autenticado.

```
src > controllers > tasks.controller.js > [o] getTask > [o] result
1 import {pool} from '../db.js'
2
3 export const getAllTasks = async (req, res, next) => {
4     console.log(req.userId)
5     const result = await pool.query('select * from task')
6     return res.json(result.rows)
7 }
```

Entonces antes de hacer cualquier petición en nuestro archivo de rutas tasks.routes.js debemos preguntar primero si esta autenticado

```
src > routes > task.routes.js > ...
7  import { isAuthenticated } from "../middlewares/auth.middleware.js";
8
9
10 const router = Router()
11
12 router.get('/tasks', isAuthenticated, getAllTasks)
13
14 router.get('/tasks/:id', isAuthenticated, getTask)
15
16 router.post('/tasks', isAuthenticated, createTask)
17
18 router.put('/tasks/:id', isAuthenticated, updateTask)
19
20 router.delete('/tasks/:id', isAuthenticated, deleteTask)
21
22 export default router
```

En las rutas de autenticación solo el profile debe preguntar si esta autenticado

```
src > routes > task.routes.js > ...
7  import { isAuthenticated } from "../middlewares/auth.middleware.js";
8
9
10 const router = Router()
11
12 router.get('/tasks', isAuthenticated, getAllTasks)
13
14 router.get('/tasks/:id', isAuthenticated, getTask)
15
16 router.post('/tasks', isAuthenticated, createTask)
17
18 router.put('/tasks/:id', isAuthenticated, updateTask)
19
20 router.delete('/tasks/:id', isAuthenticated, deleteTask)
21
22 export default router
```

Login

Hagamos el inicio de sesión

```
src > controllers > auth.controller.js > singin
6  export const singin = async (req,res)=>{
7    const { email, password } = req.body
8    const result = await pool.query('select * from users where email = $1',[email])
9    if(result.rowCount == 0){
10      return res.status(400).json({
11        message: 'El correo no esta registrado'
12      })
13    }
14    const validPassword = await bcrypt.compare(password, result.rows[0].password)
15    if(!validPassword){
16      return res.status(400).json({
17        message: 'La contraseña es incorrecta'
18      })
19    }
20
21    const token = await createAccessToken({ id: result.rows[0].id })
22    res.cookie("token", token, {
23      httpOnly: true,
24      sameSite: 'none',
25      maxAge: 24 * 60 * 60 * 1000 //1 dia
26    })
27
28  }

```

Cerrar sesión

```
src > controllers > auth.controller.js > singout
```

```
58  export const singout = (req,res)=>{
59    res.clearCookie('token')
60    res.sendStatus(200)
61 }
```

Profile

Por ejemplo si mandamos a registrar un usuario y no colocamos algún campo que es obligatorio podemos mandar un error

```
src > controllers > auth.controller.js > singup
30 export const singup = async (req, res, next) => {
31   const {name, email, password} = req.body
32   try {
33     const hashedPassword = await bcrypt.hash(password, 10)
34     console.log(hashedPassword)
35     const result = await pool.query
36     (
37       'insert into users(name, email, password) values($1,$2,$3) Returning *',
38       [name, email, hashedPassword]
39     );
40
41     const token = await createAccessToken({ id: result.rows[0].id })
42     res.cookie("token", token, [
43       httpOnly: true,
44       sameSite: 'none',
45       maxAge: 24 * 60 * 60 * 1000 //1 dia
46     ])
47     return res.json(result.rows[0])
48     //return res.json({"token": token})
49   } catch (error) {
50     if(error.code === '23505'){
51       return res.status(400).json({
52         message: 'El email ya esta registrado'
53       })
54     }
55   }
56 }
```

Ahora si miremos como traer el profile del usuario registrado

```
src > controllers > auth.controller.js > singup
64  export const profile = async(req,res)=>{
65    const result = await pool.query('select * from users where id = $1', [req.userId])
66    return res.json(result.rows[0])
67  }
68 }
```

Relación entre usuarios y tareas

Como relacionar los datos de un usuario con una tarea, entonces vamos a agregar unos campos a las tablas

```

database > init.sql
1  create table task(
2    id serial primary key,
3    title varchar(255) unique not null,
4    description varchar(255)
5  );
6
7  alter table task add column user_id integer references users(id)
8
9  create table users(
10   id serial primary key,
11   name varchar(255) not null,
12   password varchar(255) not null,
13   email varchar(255) unique not null,
14   created_at timestamp default CURRENT_TIMESTAMP,
15   updated_at timestamp default CURRENT_TIMESTAMP
16 )
17
18 alter table users add column gravatar varchar(255)

```

Vamos a hacer que cada usuario tenga una tarea a cargo, porque como lo teníamos antes podíamos ver todas las tareas y ahora estamos asignando la tarea a un usuario

```

src > controllers > tasks.controller.js > createTask > result
1 import {pool} from '../db.js'
2
3 export const getAllTasks = async (req, res, next) => {
4   console.log(req.userId)
5   const result = await pool.query('select * from task where user_id = $1', [req.userId])
6   return res.json(result.rows)
7 }
8
9 > export const getTask = async (req, res) => {...}
10
11
12 export const createTask = async (req, res, next) => {
13   const { title, description, user_id } = req.body
14   try {
15     const result = await pool.query(`insert into task (title, description, user_id) values ($1, $2, $3) returning *`, [title, description, req.userId])
16     res.json(result.rows[0])
17   } catch (error) {
18     console.log(error)
19     if(error.code === "23505"){
20       return res.status(409).json({
21         message: 'Ya existe una tarea con este título'
22       })
23     }
24   }
25   next(error)
26 }
27
28
29
30
31
32
33
34
35
36

```

Pero también podemos hacer que se nos guarde en nuestro usuario su imagen o una imagen desde gravatar

```
src > controllers > auth.controller.js > singup > result
4 import md5 from 'md5'
5
6
7 > export const singin = async (req,res)=>{ ...
29 }
30
31 export const singup = async (req,res, next)=>{
32     const {name, email, password} = req.body
33     try {
34         const hashedPassword = await bcrypt.hash(password, 10)
35         const gravatar = `https://www.gravatar.com/avatar/${md5(email)}`
36         console.log(hashedPassword)
37         const result = await pool.query
38         (
39             'insert into users(name, email, password, gravatar) values($1,$2,$3,$4) Returning *',
40             [name, email, hashedPassword, gravatar]
41         );

```

FrontEnd setup

Vamos a crear nuestro proyecto con vite, dentro de la carpeta Task que es donde tenemos nuestro Frontend

```
● danielquintero@MacBook-Air-de-Daniel Taks % sudo npm create vite
✓ Project name: ... frontend
✓ Select a framework: > React
✓ Select a variant: > JavaScript

Scaffolding project in /Users/danielquintero/Desktop/Proyectos 2024/Vite/Taks/frontend...

Done. Now run:

  cd frontend
  npm install
  npm run dev
```

Ahora hacemos la instalación, recuerde entrar a la carpeta frontend con el comando cd frontend

```
● danielquintero@MacBook-Air-de-Daniel frontend % sudo npm install
```

Y por ultimo ejecutamos el comando npm run dev

```
○ danielquintero@MacBook-Air-de-Daniel frontend % sudo npm run dev
```

```
> frontend@0.0.0 dev
> vite
```

```
VITE v5.2.11 ready in 310 ms

→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

Eliminemos todo el contenido de la pagina App.jsx y solo colocamos un hello world

```
frontend > src > App.jsx > ...
1
2  export default function App() {
3    return (
4      <div>hello world</div>
5    )
6 }
```

```
frontend > src > index.css > body
1 body{
2   background: #101010;
3   color: white;
4 }
```

Eliminamos el archivo App.css, vamos a instalar el paquete react router dom para la creación de las paginas

```
● danielquintero@MacBook-Air-de-Daniel Taks % cd frontend
● danielquintero@MacBook-Air-de-Daniel frontend % sudo npm i react-router-dom
```

Después de instalado lo vamos a importar en nuestro App.jsx para que todas las páginas se llamen por medio de este paquete

```
frontend > src > App.jsx > App
1   import { BrowserRouter } from 'react-router-dom'
2
3   export default function App() {
4     return [
5       <BrowserRouter></BrowserRouter>
6     ]
7   }
```

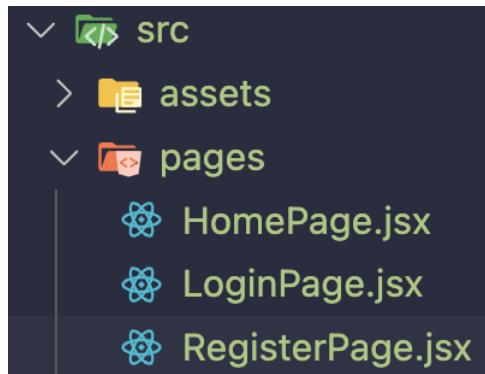
Pero si queremos que sea global lo mejor es que se haga desde el main.jsx

```
frontend > src > main.jsx
1   import React from 'react'
2   import ReactDOM from 'react-dom/client'
3   import { BrowserRouter } from 'react-router-dom'
4   import App from './App.jsx'
5   import './index.css'
6
7   ReactDOM.createRoot(document.getElementById('root')).render([
8     <React.StrictMode>
9       <BrowserRouter>
10         <App />
11       </BrowserRouter>
12     </React.StrictMode>,
13   ])
```

Y creamos una ruta en el App.jsx

```
frontend > src > App.jsx > App
1   import { Routes, Route } from 'react-router-dom'
2
3   export default function App() {
4     return [
5       <Routes>
6         <Route path='/' element={<h1>Home</h1>} />
7       </Routes>
8     ]
9   }
```

Pero como lo hicimos anteriormente no es la forma correcta porque no nos permite manejar componentes, lo mejor es crear el componente que va manejar las páginas, entonces creamos dentro del src una carpeta llamada pages y adentro los archivos que van a manejar las páginas de nuestra aplicación



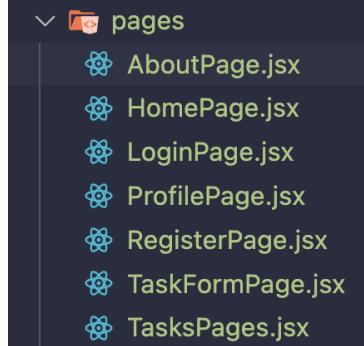
En cada pagina hacemos que se muestre un componente

```
frontend > src > pages > LoginPage.jsx > ...
1
2 export default function Logi
3   return (
4     <div>LoginPage</div>
5   )
6 }
7

frontend > src > pages > RegisterPage.jsx > ...
1
2 export default function Regi
3   return (
4     <div>RegisterPage</div>
5   )
6 }
7

frontend > src > pages > HomePage.jsx > ...
1
2 export default function Home
3   return (
4     <div>HomePage</div>
5   )
6 }
7
```

Vamos a agregar las paginas para cada una de las tareas y un profile y el about



Y a cada una le agrega su componente como se hizo anteriormente.

vamos a crear las rutas para cada uno de nuestros componentes

```

frontend > src > App.jsx > ...
  1 import { Routes, Route } from 'react-router-dom'
  2 import HomePage from './pages/HomePage'
  3 import AboutPage from './pages/AboutPage'
  4 import LoginPage from './pages/LoginPage'
  5 import RegisterPage from './pages/RegisterPage'
  6 import TasksPage from './pages/TasksPage'
  7 import TaskFormPage from './pages/TaskFormPage'
  8 import ProfilePage from './pages/ProfilePage'
  9
10 export default function App() {
11   return (
12     <Routes>
13       <Route path='/' element={<HomePage/>}/>
14       <Route path='/about' element={<AboutPage />}/>
15       <Route path='/login' element={< LoginPage/>}/>
16       <Route path='/register' element={<RegisterPage/>}/>
17
18       <Route path='/tasks' element={<TasksPage/>}/>
19       <Route path='/tasks/new' element={<TaskFormPage/>}/>
20       <Route path='/tasks/1/edit' element={<TaskFormPage/>}/>
21       <Route path='/profiel' element={<ProfilePage/>}/>
22     </Routes>
23   )
24 }

```

TailwindCSS

Lo primero es instalar la librería en nuestro frontend

```
danielquintero@MacBook-Air-de-Daniel frontend % sudo npm install -D tailwindcss postcss autoprefixer
```

Después ejecutamos el siguiente comando

```
● danielquintero@MacBook-Air-de-Daniel frontend % npx tailwindcss init -p  
Created Tailwind CSS config file: tailwind.config.js  
Created PostCSS config file: postcss.config.js
```

Lo único que hace es crear un archivo de configuración de tailwindcss

```
frontend > tailwind.config.js > ...  
1  /** @type {import('tailwindcss').Config} */  
2  export default {  
3    content: [  
4      './index.html',  
5      './src/**/*.{js,ts,jsx,tsx}',  
6    ],  
7    theme: {  
8      extend: {},  
9    },  
10   plugins: [],  
11 }
```

Con esto lo que hicimos es que la librería css que instalamos pueda aplicar cambios a toda la aplicación, ahora instalemos un complemento para el manejo de tailwindcss



Ahora en el index.css copiamos las directivas de la librería

```
frontend > src > index.css > body  
1  @tailwind base;  
2  @tailwind components;  
3  @tailwind utilities;  
4  
5  body{  
6    background: #101010;  
7    color: white;  
8  }
```

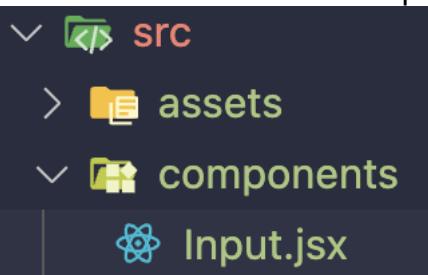
Y ya con esto tenemos la librería en nuestro proyecto y los estilos iniciales quedan aplicados

Como ejemplo de como usar la librería hagamos un pequeño ensayo

```
frontend > src > pages > RegisterPage.jsx > RegisterPage
1
2  export default function RegisterPage() {
3    return [
4      <div>
5        <h3 className="text-2xl font-bold">Register</h3>
6      </div>
7    ]
8 }
```

Formulario Registro de personas

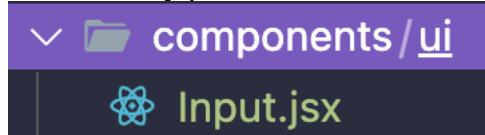
Algo que empezamos a notar es que se va repetir mucho código de la librería tailwind y para eso vamos a crear en el src una carpeta llamada components y adentro de esta un archivo Input.jsx



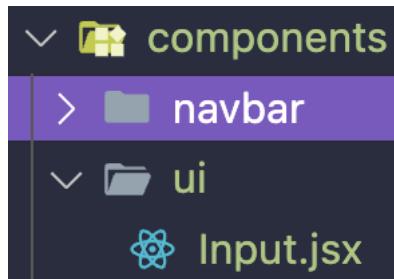
Ahora miremos como hacemos nuestro formulario de registro

```
frontend > src > pages > RegisterPage.jsx > RegisterPage
1
2  export default function RegisterPage() {
3    return [
4      <div>
5        <h3 className="text-2xl font-bold">Register</h3>
6        <form>
7          <input type="text" placeholder="Enter your full name" className="mb-2"/>
8          <input type="text" placeholder="Enter your email" />
9          <input type="text" placeholder="Enter your password" />
10         <button>register</button>
11       </form>
12     </div>
13   ]
14 }
```

Como pueden observar con el className se repite en cada uno de los input y es mejor que los estilos se manejen en un componente aparte y que sea solo llamarle cuando se necesite, dentro de componentes también creamos una carpeta llamada ui y pasemos nuestro componente Input.jsx a esta nueva carpeta



Y dentro de components voy a crear otra carpeta para el navbar ya que en el ui es para toda la aplicación y el navbar solo va estar en algunos componentes de la aplicación



Ahora podemos utilizar nuestro componente para aplicar el estilo en nuestros componentes

```
frontend > src > components > ui > Input.jsx > Input
1
2  export function Input(props) {
3    return (
4      <input type="text" className="bg-zinc-800 px-3 py-2 block my-2 w-full"
5        {...props}
6    )
7  }
8
9
10 export default Input

frontend > src > pages > RegisterPage.jsx > ...
1
2  import { Input } from '../components/ui/Input'
3
4  export default function RegisterPage() {
5    return (
6      <div>
7        <h3 className="text-2xl font-bold">Register</h3>
8        <form>
9          <Input placeholder="Enter your full name"/>
10         <Input type="email" placeholder="Enter your email"/>
11         <Input type="password" placeholder="Enter your password"/>
12         <button>register</button>
13       </form>
14     </div>
15   )
}
```

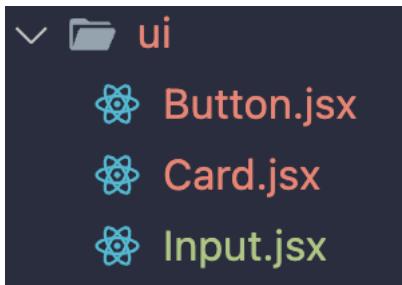
Observemos que en nuestro RegisterPage.jsx importamos el Input.jsx de nuestro componente ui, miren que dentro del componente Input utilizamos un props que es una propiedad que ponemos para utilizar cualquier tipo que deseemos utilizar con nuestros componentes

Ahora miremos que la presentación no es muy bonita entonces dentro de ui crearemos un Card.jsx

```
frontend > src > components > ui > Card.jsx > [e] default
1  export function Card( { children } ) {
2    return (
3      <div className="bg-zinc-900 p-14 rounded-md">{children}</div>
4    )
5  }
6  export default Card

frontend > src > pages > RegisterPage.jsx > ...
1  import { Input } from '../components/ui/Input'
2  import { Card } from '../components/ui/Card'
3
4  export default function RegisterPage() {
5    return (
6      <div className="h-[calc(100vh - 64px)] flex items-center justify-center mt-5">
7        <Card>
8          <h3 className="text-2xl font-bold">Register</h3>
9          <form>
10            <Input placeholder="Enter your full name"/>
11            <Input type="email" placeholder="Enter your email"/>
12            <Input type="password" placeholder="Enter your password"/>
13            <button>register</button>
14          </form>
15        </Card>
16      </div>
17    )
18 }
```

Ahora creamos el componente para el botón, dentro de la carpeta ui creamos un archivo Button.jsx



Miremos el Button.jsx

```
frontend > src > components > ui > Button.jsx > default
1  export function Button({children}) {
2    return (
3      <button
4        className="relative inline-flex
5        items-center gap-x-1.5
6        rounded-md
7        bg-indigo-500 px-3 py-1.5 text-sm
8        font-semibold text-white shadow-sm
9        hover:bg-indigo-400
10       focus-visible:outline
11       focus-visible:outline-2
12       focus-visible:outline-offset-2
13       focus-visible:outline-indigo-500
14       disabled:opacity-50 disabled:cursor-not-allowed">
15         {children}
16       </button>
17     )
18   }
19
20  export default Button
```

```
frontend > src > pages > RegisterPage.jsx > RegisterPage
1  import { Input } from '../components/ui/Input'
2  import { Card } from '../components/ui/Card'
3  import { Button } from '../components/ui/Button'
4
5  export default function RegisterPage() {
6    return [
7      <div className="h-[calc(100vh - 64px)] flex items-center justify-center mt-5">
8        <Card>
9          <h3 className="text-2xl font-bold">Register</h3>
10         <form>
11           <Input placeholder="Enter your full name"/>
12           <Input type="email" placeholder="Enter your email"/>
13           <Input type="password" placeholder="Enter your password"/>
14           <Button>Register</Button>
15         </form>
16       </Card>
17     </div>
18   ]
19 }
```

Podemos hacer que el código sea mas legible y es que en el componente RegisterPage no necesita traer todos los imports de los otros componentes

creados, yo puedo hacer que en un solo archivo se llamen y se importe en donde lo necesitemos, en la carpeta ui creamos un index.js

The screenshot shows a code editor with two tabs. The top tab is 'index.js' located at 'frontend > src > components > ui'. It contains three export statements:

```
1 export { Button } from './Button'
2 export { Card } from './Card'
3 export { Input } from './Input'
```

The bottom tab is 'RegisterPage.jsx' located at 'frontend > src > pages'. It imports 'Card', 'Button', and 'Input' from the 'ui' component directory:

```
1 import { Card, Button, Input } from "../components/ui"
```

Recuerde que vamos a estar capturando lo que se teclee en los formularios y para eso instalamos el siguiente paquete

The terminal window shows the command: `danielquintero@MacBook-Air-de-Daniel frontend % sudo npm i react-hook-form`. Below the command, the 'RegisterPage.jsx' file is shown with syntax highlighting for 'useForm' and 'register' hooks.

```
1 import { Card, Button, Input } from "../components/ui"
2 import { useForm } from "react-hook-form"
3
4 export default function RegisterPage() {
5   const { register, handleSubmit } = useForm()
6   const onSubmit = handleSubmit(data=>{
7     console.log(data)
8   })
9   return [
10     <div className="h-[calc(100vh - 64px)] flex items-center justify-center mt-5">
11       <Card>
12         <h3 className="text-2xl font-bold">Register</h3>
13         <form onSubmit={onSubmit}>
14           <Input placeholder="Enter your full name"
15             {...register('name')}
16           />
17           <Input type="email" placeholder="Enter your email"
18             {...register('email')}
19           />
20           <Input type="password" placeholder="Enter your password"
21             {...register('password')}
22           />
23           <Button>Register</Button>
24         </form>
25       </Card>
26     </div>
27   ]
28 }
```

Below the code editor, the 'Input.jsx' file is shown:

```
1 /* eslint-disable react/display-name */
2 import { forwardRef } from "react";
3
4 export const Input = forwardRef((props, ref) => {
5   return (
6     <input
7       type="text"
8       className="bg-zinc-800 px-3 py-2 block my-2 w-full"
9       ref={ref}
10      {...props}
11    />
12  );
13});
14
15 export default Input;
```

También puedo colocar que cada uno de los campos sea obligatorio

```

<Input placeholder="Enter your full name"
| {...register('name', {required: true})}
/>
<Input type="email" placeholder="Enter your email"
| {...register('email', {required: true})}
/>
<Input type="password" placeholder="Enter your password"
| {...register('password', {required: true})}
/>

```

Podemos manejar errores cuando no se llenen todos los campos

```

frontend > src > pages > RegisterPage.jsx > RegisterPage > required
4   export default function RegisterPage() {
5     const { register, handleSubmit, formState: { errors } } = useForm()
6     const onSubmit = handleSubmit(data=>{
7       console.log(data)
8     })
9     console.log(errors)
10    return (
11      <div className="h-[calc(100vh - 64px)] flex items-center justify-center mt-5">
12        <Card>
13          <h3 className="text-2xl font-bold">Register</h3>
14          <form onSubmit={onSubmit}>
15            <Input placeholder="Enter your full name"
16              {...register('name', {required: true})}
17            />
18            { errors.name && <p className="text-red-500">Name is required</p> }
19            <Input type="email" placeholder="Enter your email"
20              {...register('email', {required: true})}
21            />
22            { errors.name && <p className="text-red-500">Email is required</p> }
23            <Input type="password" placeholder="Enter your password"
24              {...register('password', {required: true})}
25            />
26            { errors.name && <p className="text-red-500">Password is required</p> }
27            <Button>Register</Button>
28          </form>
29        </Card>
30      </div>
31    )
32  }

```

Post Register

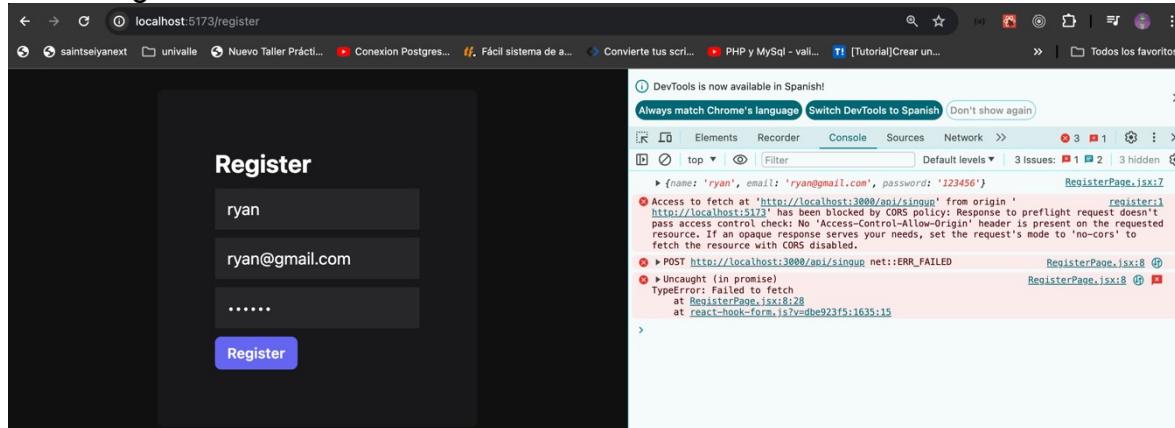
Vamos a empezar a meter información al backend para esto debemos inicializar el compilador del back

○ danielquintero@MacBook-Air-de-Daniel Taks % npm run dev

Hagamos la petición al backend

```
frontend > src > pages > RegisterPage.jsx > RegisterPage > onSubmit > handle
  4  export default function RegisterPage() {
  5    const onSubmit = handleSubmit(async(data)=>{
  6      console.log(data)
  7      const response = await fetch('http://localhost:3000/api/signup',{
  8        method: 'POST',
  9        body: JSON.stringify(data),
 10        headers: {
 11          'Content-Type':'application/json'
 12        }
 13      })
 14      const dataSignup = await response.json()
 15      console.log(dataSignup)
 16    })
 17  }
```

Ahora registremos el usuario



Resulta que ese error es por permisos de la librería cors, vayamos al app.js del backend

```
src > app.js > ...
  4  import cors from 'cors'
  5
  6  import taskRoutes from './routes/tas
  7  import authRoutes from './routes/aut
  8
  9  const app = express()
10 //middleware
11 app.use(cors({
12   origin: 'http://localhost:5173'
13 })
```

En el authcontroller.js voy a comentar el httpOnly: true para que en el localstorage se pueda almacenar la información y poder dar los permisos para que la aplicación sea segura

```

src > controllers > auth.controller.js > singin
  7  export const singin = async (req,res)=>{
 22    const token = await createAccessToken({ id: result.rows[0].id })
 23    res.cookie("token", token, {
 24      // httpOnly: true,
 25      sameSite: 'none',
 26      maxAge: 24 * 60 * 60 * 1000 //1 dia
 27    })
 28    return res.json(result.rows[0])
 29  }

```

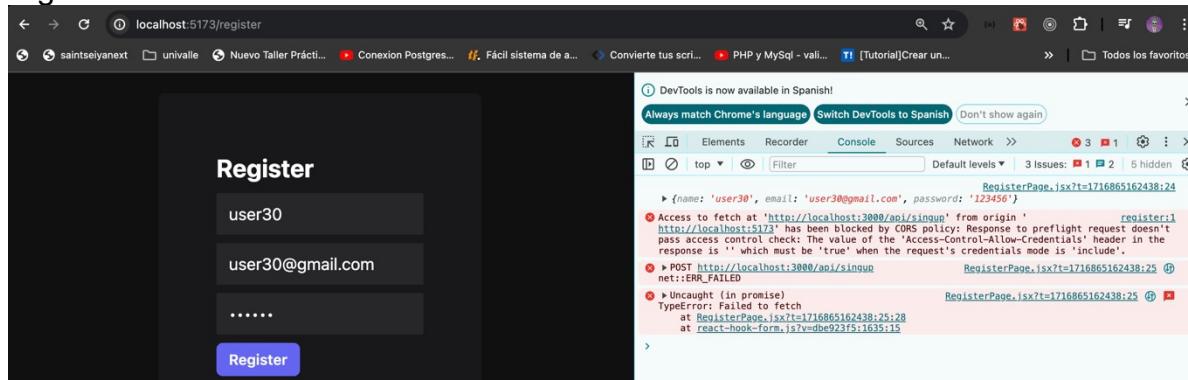
Ahora guardemos en el localstorage

```

frontend > src > pages > RegisterPage.jsx > RegisterPage > onSubmit > handle
  4  export default function RegisterPage() {
  6    const onSubmit = handleSubmit(async(data)=>{
  7      console.log(data)
  8      const response = await fetch('http://localhost:3000/api/signup',{
  9        method: 'POST',
 10       credentials: 'include',
 11       body: JSON.stringify(data),
 12       headers: {
 13         'Content-Type':'application/json'
 14       }
 15     })
 16     const dataSignup = await response.json()
 17     console.log(dataSignup)
 18   })

```

Si registramos encontramos un error que no tenemos las credenciales para registrar un usuario



Pero entonces agregamos esos permisos

```

RegisterPage.jsx > RegisterPage > onSubmit > handleSubmit() callback > response
  4  export default function RegisterPage() {
  6    const onSubmit = handleSubmit(async(data)=>{
  7      console.log(data)
  8      const response = await fetch('http://localhost:3000/api/signup',{
  9        method: 'POST',
 10       credentials: 'include',
 11       body: JSON.stringify(data),
 12       headers: {
 13         'Content-Type':'application/json',
 14         'Access-Control-Allow-Credentials':true
 15       }
 16     })

```

Y en app.js del backend también colocamos debajo del origin las credentials en true

```

src > app.js > ↗ credentials
  10 //middleware
  11 app.use(cors({
  12   origin: 'http://localhost:5173',
  13   credentials: true
  14 }))

```

Pero la cookie sigue sin guardarse entonces en el auth.controller.js vamos a colocar en la función de createAccessToken el secure: true

```

src > controllers > auth.controller.js > singin
  7 export const singin = async (req,res)=>{
  22   const token = await createAccessToken({ id: result.rows[0].id })
  23   res.cookie("token", token, {
  24     // httpOnly: true,
  25     secure: true,           ← This line is highlighted
  26     sameSite: 'none',
  27     maxAge: 24 * 60 * 60 * 1000 //1 dia
  28   })
  29   return res.json(result.rows[0])
  30 }

```

Entonces creamos un nuevo usuario y podemos observar el token guardado en la cookie del navegador

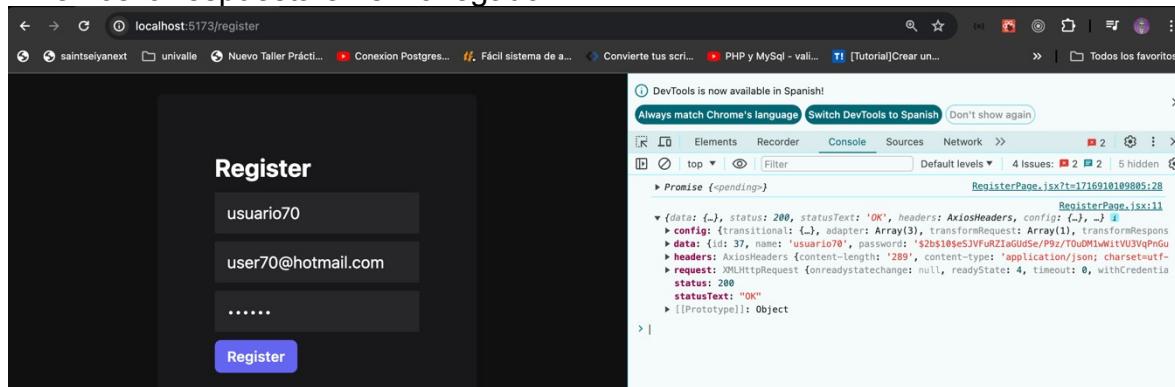
Name	Value
token	eyJhbGciOiJIUzI1Ni... eyJhbGciOiJIUzI1Ni... wiZXhwIjoxNzE2OTUyNzI1Qiw-cK8Nx1t7sBr5WqeduW3jl..._YEVjxalCTYz0OnCyw

Pero esta codificación que acabamos de desarrollar es tediosa, para poder hacer algo más estilizado y de mejor calidad vamos a instalar una librería en nuestro componente frontend llamado axios

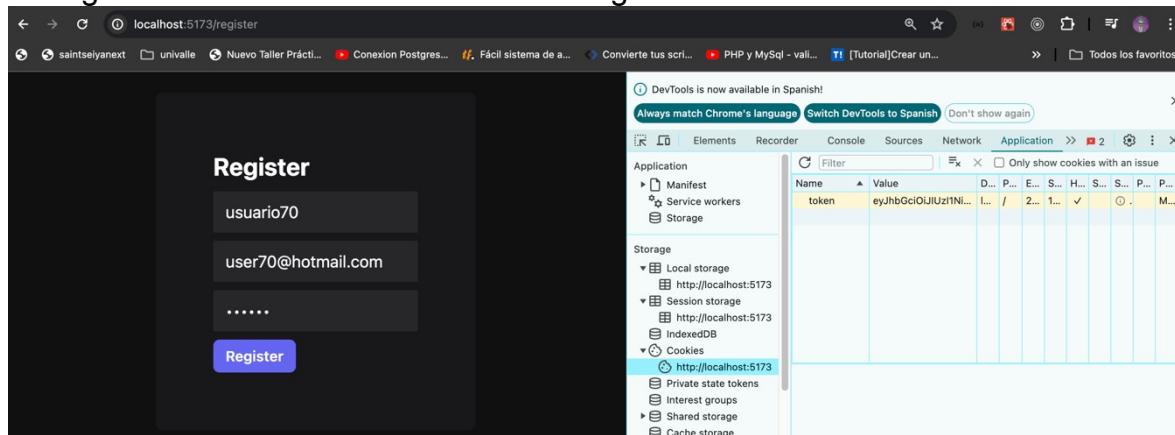
● danielquintero@MacBook-Air-de-Daniel frontend % sudo npm i axios
Miremos que nuestro código queda más legible con esta biblioteca

```
frontend > src > pages > RegisterPage.jsx > RegisterPage
  3   import axios from 'axios'
  4
  5   export default function RegisterPage() {
  6     const { register, handleSubmit, formState: { errors } } = useForm()
  7     const onSubmit = handleSubmit(async(data)=>{
  8       const res = await axios.post('http://localhost:3000/api/singup', data, {
  9         withCredentials: true
 10       })
 11       console.log(res)
 12     })
 13   }
```

Miremos la respuesta en el navegador



Y se guarda el token en la cookie del navegador



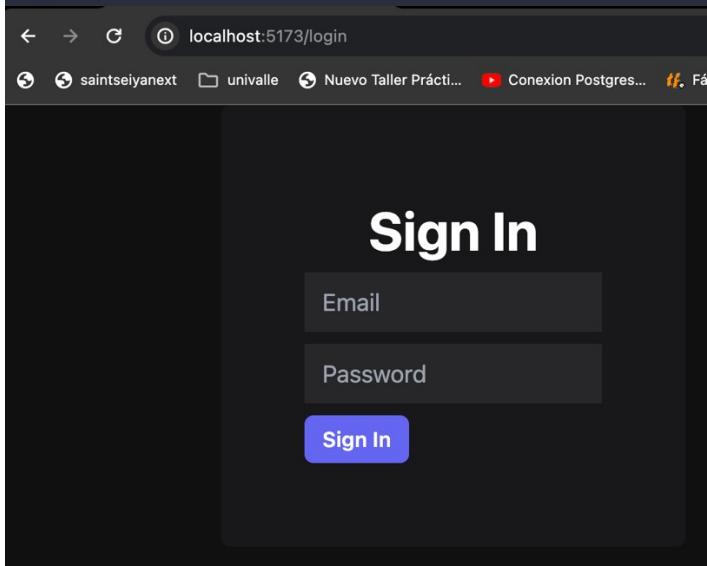
Formulario de Login

Empecemos codificando la estructura básica de nuestro login

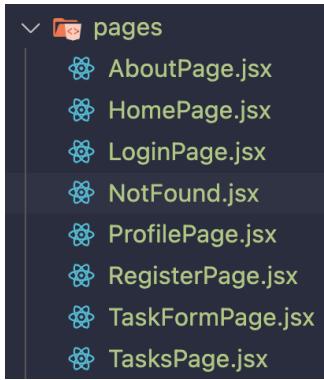
```

frontend > src > pages > LoginPage.jsx > LoginPage
1  import { Card, Input, Button } from '../components/ui'
2
3  export default function LoginPage() {
4    return [
5      <div className='h-[calc(100vh - 64px) flex justify-center items-center]>
6        <Card>
7          <h1 className='text-4xl font-bold my-2 text-center'>Sign In</h1>
8          <form>
9            <Input type="email" placeholder="Email"></Input>
10           <Input type="password" placeholder="Password"></Input>
11           <Button>
12             | Sign In
13           </Button>
14         </form>
15       </Card>
16     </div>
17   ]
18 }
19 }

```



Vamos a crear también una pagina de error cuando colocamos mal la url, en nuestro frontend creamos en pages un archivo llamado NotFound.jsx



Como queda nuestra pagina de error y recuerde que debe ser importada en el App.jsx

```
frontend > src > pages > ❓ NotFound.jsx > 🏃 default
 1 import { Link } from "react-router-dom"
 2 import { Card } from "../components/ui"
 3
 4 export function NotFound() {
 5   return (
 6     <div className='h-[calc(100vh - 64px)] flex justify-center items-center flex-col'>
 7       <Card>
 8         <h1 className="text-4xl">Page Not Found</h1>
 9         <h3 className="text-2xl">404</h3>
10         <Link to="/">Go Back to Home</Link>
11       </Card>
12     </div>
13   )
14 }
15
16 export default NotFound
```

```
frontend > src > 🌐 App.jsx > 🏁 App
```

```
 9   import NotFound from './pages/NotFound'
```

```
frontend > src > 🌐 App.jsx > 🏁 App
11   export default function App() {
23     <Route path='*' element={<NotFound/>} />
24   </Routes>
25
26 }
```

También podemos crear un componente para los label, entonces en la carpeta ui creamos un componente llamado Label.jsx

```
frontend > src > components > ui > ❓ Label.jsx > 🏁 Label
 1
 2   export function Label({ children, htmlFor }) {
 3     return (
 4       <label
 5         className="block text-sm font-medium text-gray-400"
 6         htmlFor={htmlFor}
 7       >
 8         { children }
 9       </label>
10     )
11   }
12
13 export default Label
```

Recuerde que también lo exportamos en el index.js de nuestra carpeta ui

```
frontend > src > components > ui > 🏁 index.js
 1   export { Button } from './Button'
 2   export { Card } from './Card'
 3   export { Input } from './Input'
 4   export { Label } from './Label'
```

Ahora usemos el componente Label en nuestro LoginPage.jsx

```

frontend > src > pages > LoginPage.jsx > LoginPage
  1 import { Card, Input, Button, Label } from '../components/ui'
  2 import { Link } from 'react-router-dom'
  3
  4 export default function LoginPage() {
  5   return [
  6     <div className='h-[calc(100vh - 64px)] my-5 flex justify-center items-center'>
  7       <Card>
  8         <h1 className='text-4xl font-bold my-2 text-center'>Sign In</h1>
  9         <form>
 10           <Label htmlFor="email">
 11             Email
 12           </Label>
 13           <Input type="email" placeholder="Email"></Input>
 14           <Label htmlFor="password">
 15             Password
 16           </Label>
 17           <Input type="password" placeholder="Password"></Input>
 18           <Button>
 19             Sign In
 20           </Button>
 21           <div className='flex justify-between my-4'>
 22             <p>
 23               💡 Don't have an account?
 24             </p>
 25             <Link to="/register" className='font-bold'>Register</Link>
 26           </div>
 27         </form>
 28       </Card>
 29     </div>
 30   ]
 31 }
 32 }

```

En el RegisterPage.jsx voy a colocar también un enlace que te lleve al inicio de sesión

```

frontend > src > pages > RegisterPage.jsx > RegisterPage
  6 export default function RegisterPage() {
  7   <Button>Register</Button>
  8   <div className='flex justify-between my-4'>
  9     <p>
 10       💡 Already have an account?
 11     </p>
 12     <Link to="/login" className='font-bold'> Login</Link>
 13   </div>
 14 }
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38

```

Ahora en el LoginPage.jsx vamos a capturar lo que enviamos por el formulario, recuerde que importamos el useForm, creamos dos constantes una para el manejo de obligatoriedad de los datos en el formulario

```
frontend > src > pages > 🗂 LoginPage.jsx > LoginPage
  3 import { useForm } from 'react-hook-form'
  4
  5 export default function LoginPage() {
  6   const { register, handleSubmit } = useForm()
  7
  8   const onSubmit = handleSubmit((data)=>{
  9     console.log(data)
 10   })
 11   return (
 12     <div className='h-[calc(100vh - 64px)] my-5 flex justify-center items-center'>
 13       <Card>
 14         <h1 className='text-4xl font-bold my-2 text-center'>Sign In</h1>
 15         <form onSubmit={onSubmit}>
 16           <Label htmlFor="email">
 17             Email
 18           </Label>
 19           <Input type="email" placeholder="Email"
 20             { ...register('email',{
 21               required: true
 22             })
 23           }
 24         />
 25           <Label htmlFor="password">
 26             Password
 27           </Label>
 28           <Input type="password" placeholder="Password"
 29             { ...register('password',{
 30               required: true
 31             })
 32           }
 33         />
```

Post Login

Vamos a enviar los datos al backend, recuerde que utilizamos la librería axios, recuerde que también se debe colocar el withCredentials en true

```

frontend > src > pages > LoginPage.jsx > LoginPage > onSubmit > handleSubmit() callback
4   import axios from 'axios'
5
6   export default function LoginPage() {
7     const { register, handleSubmit } = useForm()
8
9     const onSubmit = handleSubmit(async(data)=>{
10       console.log(data)
11       const res = await axios.post('http://localhost:3000/api/singin', data, [
12         withCredentials: true
13       ])
14       console.log(res)
15     })
16     return (

```

Pero cuando iniciamos la sesión no se guarda el token en la cookie del navegador, entonces en el auth.controller.js en el singin el secure en true

```

src > controllers > auth.controller.js > singin
7   export const singin = async (req,res)=>{
22     const token = await createAccessToken({ id: result.rows[0].id })
23     res.cookie("token", token, [
24       // httpOnly: true,
25       secure: true,
26       sameSite: 'none',
27       maxAge: 24 * 60 * 60 * 1000 //1 dia
28     ])
29     return res.json(result.rows[0])
30   }

```

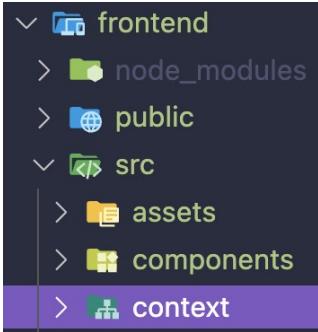
Nos logueamos y vemos que el token se registra en las cookies del navegador

The screenshot shows a browser window with a sign-in form on the left and the Chrome DevTools Application tab on the right. The sign-in form has fields for Email (user70@gmail.com) and Password, and a Sign In button. The DevTools Application tab shows a table of cookies. One cookie is selected, named 'token' with a value of 'eyJhbGciOiJIUzI1Ni...'. The table includes columns for Name, Value, and various expiration and persistence properties.

Name	Value	D...	P...	E...	S...	H...	S...	P...	P...
token	eyJhbGciOiJIUzI1Ni...								

Auth Context

Crear un contexto de todos los componentes, vamos a crear una carpeta en el src del frontend llamada context en react es un componente



Dentro de la carpeta vamos a crear un archivo AuthContext.jsx

```
frontend > src > context > AuthContext.jsx > ...
1 import { createContext } from 'react'
2
3 const context = createContext()
4
5 export function AuthProvider({ children }) {
6     return <context.Provider>
7     |
8     |     { children }
9     </context.Provider>
}
```

El mejor lugar para llamar nuestro contexto es en el App.jsx, entonces todas las rutas las vamos a meter dentro de nuestro AuthContext

```
frontend > src > App.jsx > App
10 import AuthContext from './context/AuthContext'
11
12 export default function App() {
13     return (
14         <AuthContext>
15             <Routes>
16                 <Route path='/' element={<HomePage/>} />
17                 <Route path='/about' element={<AboutPage />} />
18                 <Route path='/login' element={< LoginPage/>} />
19                 <Route path='/register' element={<RegisterPage/>} />
20
21                 <Route path='/tasks' element={<TasksPage/>} />
22                 <Route path='/tasks/new' element={<TaskFormPage/>} />
23                 <Route path='/tasks/1/edit' element={<TaskFormPage/>} />
24                 <Route path='/profile' element={<ProfilePage/>} />
25                 <Route path='*' element={<NotFound/>} />
26             </Routes>
27         </AuthContext>
28     )
29 }
```

Pero queremos saber si el usuario esta logueado vamos a mover el AuthContext al main.jsx, dejemos el archivo App.jsx (quitar el Autcontext) como estaba y vamos a englobar toda la app

```

frontend > src > main.jsx
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import { BrowserRouter } from 'react-router-dom'
4 import AuthContext from './context/AuthContext'
5 import App from './App.jsx'
6 import './index.css'
7
8 ReactDOM.createRoot(document.getElementById('root')).render([
9   <React.StrictMode>
10    <BrowserRouter>
11      <AuthContext>
12        <App />
13      </AuthContext>
14    </BrowserRouter>
15  </React.StrictMode>,
16])

```

El AuthContext nos queda

```

frontend > src > context > AuthContext.jsx > ...
1 import { createContext, useState } from 'react'
2
3 export const AuthContext = createContext()
4
5 export function AuthProvider({ children }) {
6   const [user, setUser] = useState(null)
7   const [isAuth, setIsAuth] = useState(false)
8   const [errors, setErrors] = useState(false)
9   return <AuthContext.Provider value={{
10     user,
11     isAuth,
12     errors
13   }}>
14   { children }
15 </AuthContext.Provider>
16 }

```

Y si queremos mostrar que nos esta capturando algún componente lo podemos importar en cualquiera, nuestro main.jsx hace un llamado al AuthProvider de nuestro AuthContext.jsx

```

frontend > src > main.jsx
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import { BrowserRouter } from 'react-router-dom'
4 import { AuthProvider } from './context/AuthContext'
5 import App from './App.jsx'
6 import './index.css'
7
8 ReactDOM.createRoot(document.getElementById('root')).render(
9   <React.StrictMode>
10    <BrowserRouter>
11      <AuthProvider>
12        <App />
13      </AuthProvider>
14    </BrowserRouter>
15  </React.StrictMode>,
16 )

```

```
frontend > src > pages > HomePage.jsx > ...
1 import { useContext } from "react"
2 import { AuthContext } from '../context/AuthContext'
3 export default function HomePage() {
4   const data = useContext(AuthContext)
5   console.log(data)
6   return (
7     | <div>HomePage</div>
8   )
9 }
```

Pero es mas fácil crear un hook personalizado en nuestro AuthContext

```
frontend > src > context > AuthContext.jsx > ...
1 import { createContext, useState, useContext } from 'react'
2
3 export const AuthContext = createContext()
4
5 export const useAuth = () => {
6   const context = useContext(AuthContext)
7   if(!context){
8     | throw new Error('UseAuth must be used within anAuthProvider')
9   }
10  return context
11 }
12
13 export function AuthProvider({ children }) {
14   const [user, setUser] = useState(null)
15   const [isAuth, setIsAuth] = useState(false)
16   const [errors, setErrors] = useState(false)
17   return <AuthContext.Provider value={{|
18     | user,
19     | isAuth,
20     | errors
21   }}>
22   | { children }
23 </AuthContext.Provider>
24 }
```

```
frontend > src > pages > HomePage.jsx > ...
1 import { useAuth } from '../context/AuthContext'
2 export default function HomePage() {
3   const data = useAuth()
4   console.log(data)
5   return (
6     | <div>HomePage</div>
7   )
8 }
```

Login y Registro

Como mejorar las rutas de login y de register aplicando un contexto, en el registerPage vamos a quitar las siguientes líneas como se ve a continuación

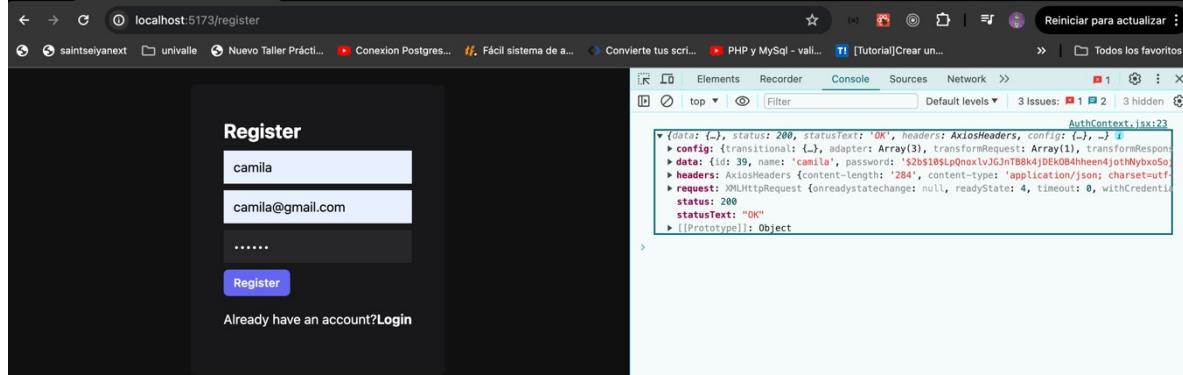
```
frontend > src > pages > RegisterPage.jsx > RegisterPage
1  import { Card, Button, Input } from "../components/ui"
2  import { useForm } from 'react-hook-form'
3  import { Link } from "react-router-dom"
4  import { useAuth } from "../context/AuthContext"
5
6  export default function RegisterPage() {
7    const { register,
8      |   |   | handleSubmit,
9      |   |   | formState: {errors}
10     } = useForm()
11    const { singup } = useAuth()
12
13    const onSubmit = handleSubmit(async(data)=>{
14      |   await singup(data)
15    })
}
```

```

frontend > src > context > AuthContext.jsx > AuthProvider
1   import { createContext, useState, useContext } from 'react'
2   import axios from 'axios'
3
4   export const AuthContext = createContext()
5
6   export const useAuth = () => {
7     const context = useContext(AuthContext)
8     if(!context){
9       throw new Error('UseAuth must be used within anAuthProvider')
10    }
11    return context
12  }
13
14  export function AuthProvider({ children }) {
15    const [user, setUser] = useState(null)
16    const [isAuth, setIsAuth] = useState(false)
17    const [errors, setErrors] = useState(false)
18
19    const signup = async(data) => {
20      const res = await axios.post('http://localhost:3000/api/signup', data, {
21        withCredentials: true
22      })
23      console.log(res)
24    }
25
26    return <AuthContext.Provider value={{{
27      user,
28      isAuth,
29      errors,
30      signup
31    }}}>
32      { children }
33    </AuthContext.Provider>
34  }

```

Ahora ingresemos a la url <http://localhost:5173/register>



También podemos hacer que cada vez que nos registremos nos redireccione al perfil

```

frontend > src > pages > RegisterPage.jsx > RegisterPage
1  import { Card, Button, Input } from "../components/ui"
2  import { useForm } from 'react-hook-form'
3  import { Link, useNavigate } from "react-router-dom"
4  import { useAuth } from "../context/AuthContext"

5
6  export default function RegisterPage() {
7    const { register,
8      | | | handleSubmit,
9      | | | formState: {errors}
10   } = useForm()
11  const { singup } = useAuth()
12  const navigate = useNavigate()

13
14  const onSubmit = handleSubmit(async(data)=>{
15    await singup(data)
16    navigate('/profile')
17 })

```

Estamos mostrando los datos pero debemos guardarlos entonces esto lo hacemos en el AuthContext.jsx

```

frontend > src > context > AuthContext.jsx > AuthProvider > singup
14  export function AuthProvider({ children }){
15    const [user, setUser] = useState(null)
16    const [ isAuthenticated, setIsAuthenticated ] = useState(false)
17    const [ errors, setErrors ] = useState(false)

18
19    const singup = async(data) => [
20      const res = await axios.post('http://localhost:3000/api/singup', data,{
21        withCredentials: true
22      })
23      console.log(res.data)
24      setUser(res.data)
25    ]

```

Accedamos a los datos desde el ProfilePage.jsx

```

frontend > src > pages > ProfilePage.jsx > ProfilePage
1  import { useAuth } from '../context/AuthContext'

2
3  export default function ProfilePage() {
4    const { user } = useAuth()
5    return (
6      <div>
7        | {JSON.stringify(user)}
8      </div>
9    )
10 }

```

The screenshot shows a browser window with the URL `localhost:5173/profile`. The page content is a JSON object:

```
{"id":41,"name":"mario bros","password":"$2b$10$3PETZ7yiqh88M6darySq.yx5SqA.qL2o0bF0GRju94vSRFCVBCA2","email":"mariob@gmail.com","created_at":"2024-06-21T03:58:25.406Z","updated_at":"2024-06-21T03:58:25.406Z","gravatar":"https://www.gravatar.com/avatar/5da87b0221de978e323dc9fdd0959c12"}
```

Below the JSON, the browser's developer tools are open, specifically the Components tab. It shows a tree structure of a React component named `AuthProvider`. The `AuthProvider` component has a prop named `children` which contains a state object with the same properties as the JSON above. The state object is expanded to show its values.

```
AuthProvider
  props
    > children: ($stypeof: Symbol/react.element), _owner: n...
    new entry: ""
  Collapse prop value
  State: {created_at: "2024-06-21T03:58:25.406Z", em...
    id: 41
    name: "mario bros"
    password: "$2b$10$3PETZ7yiqh88M6darySq.yx5SqA.qL2o...
    email: "mariob@gmail.com"
    created_at: "2024-06-21T03:58:25.406Z"
    updated_at: "2024-06-21T03:58:25.406Z"
```

Ahora podemos hacer lo mismo en el login

```

frontend > src > context > 🌐 AuthContext.jsx > ⚡ AuthProvider
14  export function AuthProvider({ children }) {
15    const [user, setUser] = useState(null)
16    const [isAuth, setIsAuth] = useState(false)
17    const [errors, setErrors] = useState(false)
18
19    const singin = async(data) => {
20      const res = await axios.post('http://localhost:3000/api/singin', data, {
21        withCredentials: true
22      })
23      console.log(res.data)
24      setUser(res.data)
25    }
26
27    const singup = async(data) => {
28      const res = await axios.post('http://localhost:3000/api/singup', data, {
29        withCredentials: true
30      })
31      console.log(res.data)
32      setUser(res.data)
33    }
34
35    return <AuthContext.Provider value={{
36      user,
37      isAuth,
38      errors,
39      singup,
40      singin
41    }}>
42    { children }
43  </AuthContext.Provider>
44 }

```

```

frontend > src > pages > 🌐 LoginPage.jsx > ⚡ onSubmit > ⚡ handle
1  import { Card, Input, Button, Label } from '../components/ui'
2  import { Link } from 'react-router-dom'
3  import { useForm } from 'react-hook-form'
4  import { useAuth } from '../context/AuthContext'
5
6  export default function LoginPage() {
7    const { register, handleSubmit } = useForm()
8    const { singin } = useAuth()
9
10   const onSubmit = handleSubmit(async(data)=>{
11     await singin(data)
12   })

```

Ya podemos iniciar la sesión con algún usuario creado, pero también queremos que cuando iniciemos sesión nos vayamos al perfil

```

frontend > src > pages > 🌐 LoginPage.jsx > ⚡ onSubmit > ⚡ handle
1  import { Card, Input, Button, Label } from '../components/ui'
2  import { Link, useNavigate } from 'react-router-dom'
3  import { useForm } from 'react-hook-form'
4  import { useAuth } from '../context/AuthContext'
5
6  export default function LoginPage() {
7    const { register, handleSubmit } = useForm()
8    const { singin } = useAuth()
9    const navigate = useNavigate()
10
11   const onSubmit = handleSubmit(async(data)=>{
12     await singin(data)
13     navigate('/profile')
14   })

```

Backend Errors

Como manejar errores cuando el usuario coloca credenciales invalidas, dentro del backend en el src → schemas vamos a crear un archivo task.schema.js

```
src > schemas > task.schema.js > ...
1 import { z } from 'zod'
2
3 export const createTaskSchema = z.object({
4   title: z.string().min(1).max(255),
5   description: z.string().min(1).max(255).nullable()
6 })
```

Recuerde que estamos trabajando en el backend por lo tanto vamos a crear otro middleware para el manejo de las validaciones

```
✓ middlewares
  └── auth.middleware.js
  └── validate.middleware.js U

src > middlewares > validate.middleware.js > validateSchema > <function>
1 export const validateSchema = (schema) => async (req, res, next) =>[
2   try {
3     await schema.parse(req.body)
4     next()
5   } catch (error) {
6     console.log(error)
7     return res.status(400).json({ error: error.message })
8   }
9 ]
```

Debemos importarlo en las rutas en tasks.routes.js

```
src > routes > task.routes.js > router
8 import { validateSchema } from "../middlewares/validate.middleware.js";
9 import { createTaskSchema } from "../schemas/task.schema.js";
...
src > routes > task.routes.js > router
17 router.post('/tasks', isAuthenticated, validateSchema(createTaskSchema), createTask)
18
```

Hacemos una prueba desde el postman, la primera prueba es mandar un objeto vacío donde nos sale el mensaje de error no estas autorizado

POST | http://localhost:3000/api/tasks

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body (Pretty, Raw, Preview, JSON)

```

1 []
2 ...
3 []

```

Body Cookies Headers (10) Test Results

Status: 401 Unauthorized Time: 8 ms Size: 384 B

```

1 []
2 "message": "No estas autorizado"
3 []

```

Vamos a loguearnos primero

POST | http://localhost:3000/api/singin

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Body (Pretty, Raw, Preview, JSON)

```

1 {}
2 ... "email": "camila@gmail.com",
3 ... "password": "123456"
4 {}

```

Vamos a corregir un error en el validate.middleware.js no es errors es error

```

src > middlewares > validate.middleware.js > validateSchema > <function>
1   export const validateSchema = (schema) => async (req, res, next) =>{
2     try {
3       await schema.parse(req.body)
4       next()
5     } catch (error) {
6       console.log(error)
7       return res.status(400).json({ error: error.message })
8     }
9   }

```

Si enviamos una tarea vacía nos genera un error pero no nos detecta que nos logueamos a lo cual en el auth.controller.js en el createAccessstoken vamos a comentar la línea de secure: true

```

src > controllers > auth.controller.js > singin
1   export const singin = async (req, res) => {
2     const token = await createAccessToken({ id: result.rows[0].id })
3     res.cookie("token", token, {
4       // httpOnly: true,
5       // secure: true,
6       sameSite: 'none',
7       maxAge: 24 * 60 * 60 * 1000 //1 dia
8     })
9     return res.json(result.rows[0])
10   }

```

Cuando enviamos el objeto vacío o con datos erróneos se muestra el siguiente error

```

POST http://localhost:3000/api/tasks
Body (raw JSON)
{
}

```

Status: 400 Bad Request Time: 6 ms Size: 737 B

```

1
2
3
"error": "[\n    {\n        "code": "invalid_type",\n        "expected": "string",\n        "received": "\nundefined",\n        "path": [\n            "title"\n        ],\n        "message": "Required"\n    },\n    {\n        "code": "invalid_type",\n        "expected": "string",\n        "received": "\nundefined",\n        "path": [\n            "description"\n        ],\n        "message": "Required"\n    }\n]"

```

En el task.schema.js podemos mandar mensajes de error personalizados

```

src > schemas > task.schema.js > [e] createTaskSchema > ↗ description > ↗ invalid_type_error
1 import { z } from 'zod'
2
3 export const createTaskSchema = z.object({
4     title: z.string({
5         required_error: "El título es requerido",
6         invalid_type_error: "El título debe ser un texto"
7     }).min(1).max(255),
8     description: z.string({
9         required_error: "La descripción es requerida",
10        invalid_type_error: "La descripción debe ser un texto"
11    }).min(1).max(255).nullable()
12 })

```

Ahora en el validate.middleware.js podemos mejorar el mensaje de error

```

src > middlewares > validate.middleware.js > ...
1 export const validateSchema = (schema) => async (req, res, next) =>{
2     try {
3         await schema.parse(req.body)
4         next()
5     } catch (error) {
6         console.log(error.errors)
7         if(Array.isArray(error.errors)){
8             return res.status(400).json(error.errors.map(error => error.message))
9         }
10    return res.status(400).json(error.message)
11 }
12 }

```

Enviamos una tarea vacía y mostremos el error

POST | http://localhost:3000/api/tasks

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body (Pretty Raw Preview JSON)

```

1 []
2 ...
3 []

```

Status: 400 Bad Request Time: 29 ms Size: 407 B

```

1 []
2   "El titulo es requerido",
3   "La descripción es requerida"
4 []

```

Recordemos que la descripción de la tarea es opcional entonces en vez de colocar nullable colocamos optional

```

src > schemas > task.schema.js > [e] createTaskSchema > ⚡ description
1 import { z } from 'zod'
2
3 export const createTaskSchema = z.object({
4   title: z.string({
5     required_error: "El titulo es requerido",
6     invalid_type_error: "El titulo debe ser un texto"
7   }).min(1).max(255),
8   description: z.string({
9     required_error: "La descripción es requerida",
10    invalid_type_error: "La descripción debe ser un texto"
11  }).min(1).max(255).optional()
12 })

```

Vamos a insertar una tarea

POST | http://localhost:3000/api/tasks

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Body (Pretty Raw Preview JSON)

```

1 []
2   "title": "Mi tarea"
3 []

```

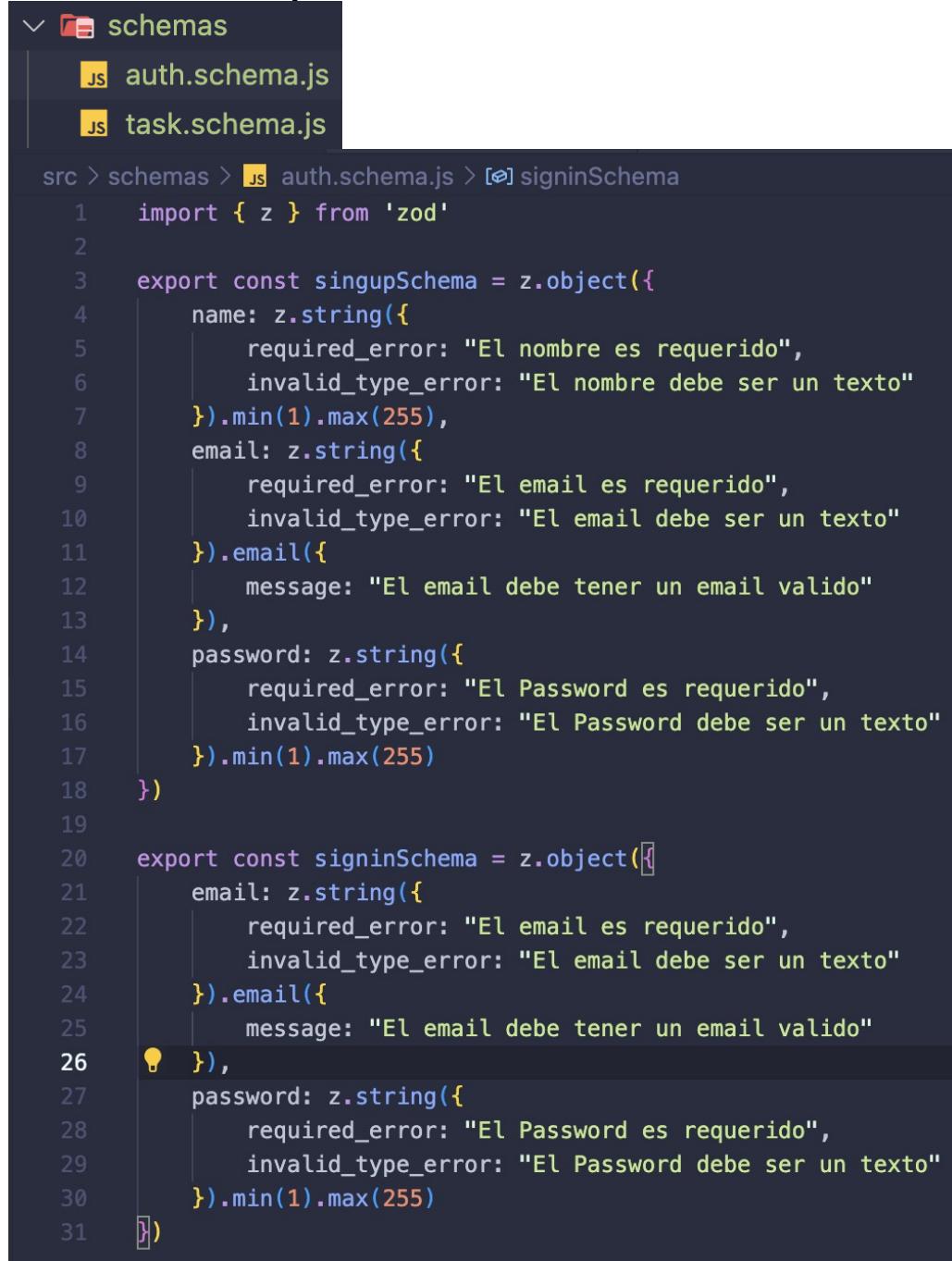
Status: 200 OK Time: 104 ms Size: 401 B

```

1 []
2   "id": 19,
3   "title": "Mi tarea",
4   "description": null,
5   "user_id": 39
6

```

Vamos a hacer las diferentes validaciones para el login, en schemas vamos a crear el auth.schema.js



```
src > schemas > auth.schema.js > [e] signinSchema
1 import { z } from 'zod'
2
3 export const singupSchema = z.object({
4     name: z.string({
5         required_error: "El nombre es requerido",
6         invalid_type_error: "El nombre debe ser un texto"
7     }).min(1).max(255),
8     email: z.string({
9         required_error: "El email es requerido",
10        invalid_type_error: "El email debe ser un texto"
11    }).email({
12        message: "El email debe tener un email valido"
13    }),
14     password: z.string({
15         required_error: "El Password es requerido",
16         invalid_type_error: "El Password debe ser un texto"
17     }).min(1).max(255)
18 })
19
20 export const signinSchema = z.object([
21     email: z.string({
22         required_error: "El email es requerido",
23         invalid_type_error: "El email debe ser un texto"
24     }).email({
25         message: "El email debe tener un email valido"
26     }),
27     password: z.string({
28         required_error: "El Password es requerido",
29         invalid_type_error: "El Password debe ser un texto"
30     }).min(1).max(255)
31 ])
```

Ahora como lo usamos, abrimos el auth.route.js

```

src > routes > auth.routes.js > ...
4 import { validateSchema } from '../middlewares/validate.middleware.js'
5 import { singupSchema, signinSchema } from '../schemas/auth.schema.js'
6
7 const router = Router()
8
9 router.post('/singin', validateSchema(signinSchema), signin)
10
11 router.post('/singup', validateSchema(singupSchema), singup)

```

Ahora vamos a probarlo, primero enviamos el objeto vacío

The screenshot shows the Postman interface. The URL is set to `http://localhost:3000/api/singup`. The method is `POST`. In the `Body` tab, the `raw` option is selected, and the JSON body is empty (represented by a single digit '1'). The response status is `400 Bad Request`, and the error message is: "El nombre es requerido", "El email es requerido", "El Password es requerido".

También voy a crear un schema para la actualización

```

src > schemas > task.schema.js > ...
14 export const updateTaskSchema = z.object({
15   title: z.string({
16     required_error: "El titulo es requerido",
17     invalid_type_error: "El titulo debe ser un texto"
18   }).min(1).max(255).optional(),
19   description: z.string({
20     required_error: "La descripción es requerida",
21     invalid_type_error: "La descripción debe ser un texto"
22   }).min(1).max(255).optional()
23 })

```

```

src > routes > task.routes.js > ...
9 import { createTaskSchema, updateTaskSchema } from '../schemas/task.schema.js'
10
11 const router = Router()
12
13 router.get('/tasks', isAuthenticated, getAllTasks)
14
15 router.get('/tasks/:id', isAuthenticated, getTask)
16
17 router.post('/tasks', isAuthenticated, validateSchema(createTaskSchema), createTask)
18
19 router.put('/tasks/:id', isAuthenticated, validateSchema(updateTaskSchema), updateTask)

```

Recordar colocar en el auth.controller.js el secure true

```
src > controllers > [JS] auth.controller.js > [✖] singin > ✎ sameSite
  7   export const singin = async (req,res)=>
 21
 22     const token = await createAccessToken({ id: result.rows[0].id })
 23     res.cookie("token", token, [
 24       // httpOnly: true,
 25       secure: true,
```

Frontend Errors

Vamos a mostrar los errores que se pueden generar desde el frontend. Vamos a abrir el AuthContext.jsx, recuerde que aca vamos a trabajar en el front. En el singin vamos a llamar el arreglo de errores

```

frontend > src > context > ✨ AuthContext.jsx > ⚡ AuthProvider
14  export function AuthProvider({ children }) {
15    const singin = async(data) =>{
16      try {
17        const res = await axios.post('http://localhost:3000/api/singin', data, {
18          withCredentials: true,
19        })
20        setUser(res.data)
21      } catch(error) {
22        console.log(res.data)
23        if(Array.isArray(error.response.data)){
24          return setErrors(error.response.data)
25        }
26        setErrors([error.response.data.message])
27      }
28    }
29  }
30}
31}
32}

```

Si queremos ver los errores debemos ir a LoginPage.jsx

```

frontend > src > pages > ✨ LoginPage.jsx > ⚡ LoginPage > 🚧 onSubmit > ⚡ handleSubmit() callback
6  export default function LoginPage() {
7    const { singin, errors } = useAuth()
8    const navigate = useNavigate()
9
10   const onSubmit = handleSubmit(async(data)=>{
11     await singin(data)
12     // navigate('/profile')
13   })
14
15   return (
16     <div className='h-[calc(100vh - 64px)] my-5 flex justify-center items-center'>
17       <Card>
18         {
19           JSON.stringify(errors)
20         }
21     </Card>
22   )
23 }
24
25
26
27
28
29
30
31
32
33
34

```

A parte de cuando nos logueamos debemos estar en true después de pasar los errores de autenticación si los hay

```

frontend > src > context > ✨ AuthContext.jsx > ⚡ AuthProvider
14  export function AuthProvider({ children }) {
15    const singin = async(data) =>{
16      try {
17        const res = await axios.post('http://localhost:3000/api/singin', data, {
18          withCredentials: true,
19        })
20        setUser(res.data)
21        setIsAuth(true)
22        return res.data
23      } catch(error) {
24        console.log(error)
25        if(Array.isArray(error.response.data)){
26          return setErrors(error.response.data)
27        }
28        setErrors([error.response.data.message])
29      }
30    }
31  }
32}
33}
34}

```

Organicemos un poco mejor los mensajes de error, pero también hicimos que nos retorne los datos del usuario autenticado

```
frontend > src > pages > LoginPage.jsx > LoginPage > onSubmit > handleSubmit() callback
  6   export default function LoginPage() {
  7     const onSubmit = handleSubmit(async(data)=>{
  8       const user = await singin(data)
  9       if(user){
 10         navigate['/profile']
 11       }
 12     })
 13     return (
 14       <div className='h-[calc(100vh - 64px)] my-5 flex justify-center items-center'>
 15         <Card>
 16           {
 17             errors && (
 18               errors.map(err => (
 19                 <p className='bg-red-500 text-white text-center'>
 20                   {err}
 21                 </p>
 22               ))
 23             )
 24           }
 25         )
 26       )
 27     }
 28   }
```

Hacemos el mismo proceso para el singup

```

frontend > src > context > ✨ AuthContext.jsx > 📂 AuthProvider > [ej] singup
14  export function AuthProvider({ children }){
36    const singup = async(data) => {
37      try {
38        const res = await axios.post('http://localhost:3000/api/singup', data, {
39          withCredentials: true
40        })
41        setUser(res.data)
42        setIsAuth(true)
43        return res.data
44      } catch (error) {
45        console.log(error)
46        if(Array.isArray(error.response.data)){
47          return setErrors(error.response.data)
48        }
49        setErrors([error.response.data.message])
50      }
51    }
52  }

frontend > src > pages > ✨ RegisterPage.jsx > 📂 RegisterPage
6  export default function RegisterPage() {
11  const { singup, errors: singupErrors } = useAuth()
12  const navigate = useNavigate()
13
14  const onSubmit = handleSubmit(async(data)=>{
15    const user = await singup(data)
16    if(user){
17      navigate('/profile')
18    }
19  })
20
21  return (
22    <div className="h-[calc(100vh - 64px)] flex items-center justify-center mt-5">
23      <Card>
24        [
25          singupErrors && (
26            singupErrors.map(err => (
27              <p className='bg-red-500 text-white text-center'>
28                {err}
29              </p>
30            )))
31        ]
32      }
33    )
34  }

```

Vamos a hacer que el usuario sea persistente, para esto debemos instalar una librería todo en la carpeta del frontend

```
● danielquintero@MacBook-Air-de-Daniel frontend % sudo npm i js-cookie
```

Después de esto vamos a llamar una librería useEffect e importamos la librería instalada js-cookie

```
frontend > src > context > AuthContext.jsx >AuthProvider > useEffect() callback >
  1  import { createContext, useState, useContext, useEffect } from 'react'
  2  import Cookie from 'js-cookie'

frontend > src > context > AuthContext.jsx > ...
  15 export function AuthProvider({ children }) {
  54   useEffect(()=>{
  55     if(Cookie.get('token')){
  56       axios
  57         .get('http://localhost:3000/api/profile',{
  58           withCredentials: true
  59         })
  60         .then((res)=>{
  61           console.log(res.data)
  62           setIsAuth(true)
  63         })
  64         .catch((err)=>{
  65           console.log(err)
  66           setUser(null)
  67           setIsAuth(false)
  68         })
  69     }
  70   })
}
```

Para no estar llamando la url en cada función lo podemos hacer en un archivo como lo vamos a hacer a continuación, dentro de la carpeta src del frontend

frontend

- > node_modules
- > public
- < src
 - < api
 - JS axios.js

```
frontend > src > api > JS axios.js > [e] default
1 import axios from "axios";
2
3 const client = axios.create({
4   baseURL: 'http://localhost:3000/api',
5   withCredentials: true
6 })
7
8 export default client|
```

En cada una de nuestras rutas es solo llamar la que necesitamos

frontend > src > context > AuthContext.jsx > AuthProvider > useEffect

```
15 export function AuthProvider({ children }) {
20   const singin = async(data) => {
21     try {
22       const res = await axios.post('/singin', data)
```

frontend > src > context > AuthContext.jsx > AuthProvider > useEffect

```
15 export function AuthProvider({ children }) {
35   const singup = async(data) => {
36     try {
37       const res = await axios.post('/singup', data)
```

frontend > src > context > AuthContext.jsx > AuthProvider > useEffect

```
15 export function AuthProvider({ children }) {
50   useEffect(()=>{
51     if(Cookie.get('token')){
52       axios
53         .get('/profile')
```

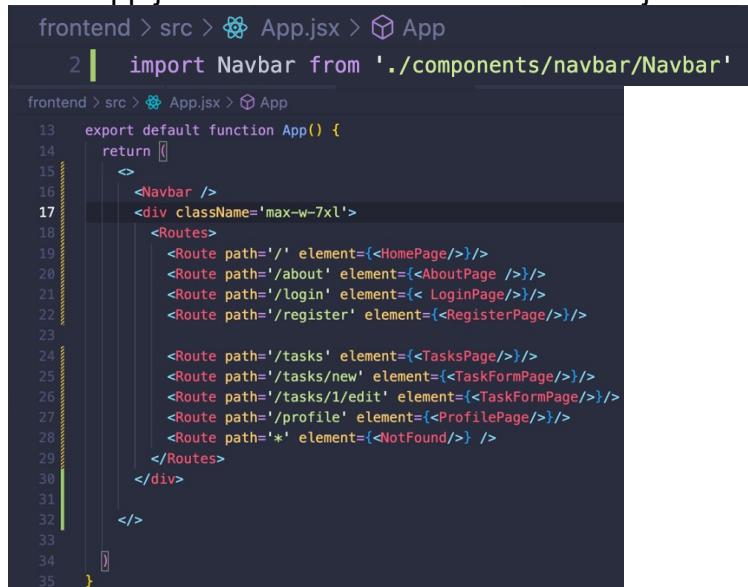
NavBar

Como hacer la navegación para navegar entre páginas, vamos a crear dentro de navbar un archivo llamado navigation.js que es donde vamos a colocar el arreglo de navegación entre las diferentes rutas



```
frontend > src > components > navbar > navigation.js > ...
1  export const navigation = [
2    {
3      name: 'Home',
4      path: '/'
5    },
6    {
7      name: 'About',
8      path: '/about'
9    },
10   {
11     name: 'Login',
12     path: '/login'
13   },
14   {
15     name: 'Register',
16     path: '/register'
17   },
18   {
19     name: 'Tasks',
20     path: '/tasks'
21   },
22   {
23     name: 'New Task',
24     path: '/tasks/new'
25   },
26   {
27     name: 'Edit Task',
28     path: '/tasks/1/edit'
29   },
30   {
31     name: 'Profile',
32     path: '/profile'
33   }
34 ]
```

En el App.jsx debo llamar mi archivo Navbar.jsx



```
frontend > src > App.jsx > App
2 import Navbar from './components/navbar/Navbar'

frontend > src > App.jsx > App
13 export default function App() {
14   return (
15     <>
16       <Navbar />
17       <div className='max-w-7xl'>
18         <Routes>
19           <Route path='/' element={<HomePage/>}/>
20           <Route path='/about' element={<AboutPage />}/>
21           <Route path='/login' element={<LoginPage/>}/>
22           <Route path='/register' element={<RegisterPage/>}/>

23           <Route path='/tasks' element={<TasksPage/>}/>
24           <Route path='/tasks/new' element={<TaskFormPage/>}/>
25           <Route path='/tasks/1/edit' element={<TaskFormPage/>}/>
26           <Route path='/profile' element={<ProfilePage/>}/>
27           <Route path='*' element={<NotFound/>} />
28         </Routes>
29       </div>
30     </>
31   
```

Pero también podemos crear un Container.jsx

