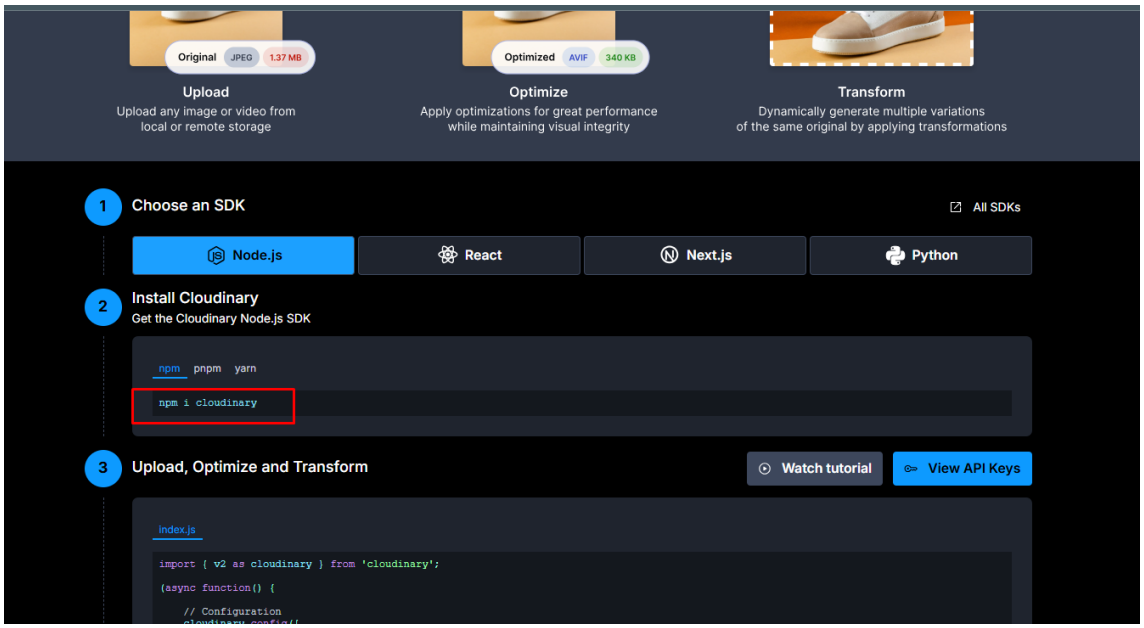
Subir Imágenes de perfil

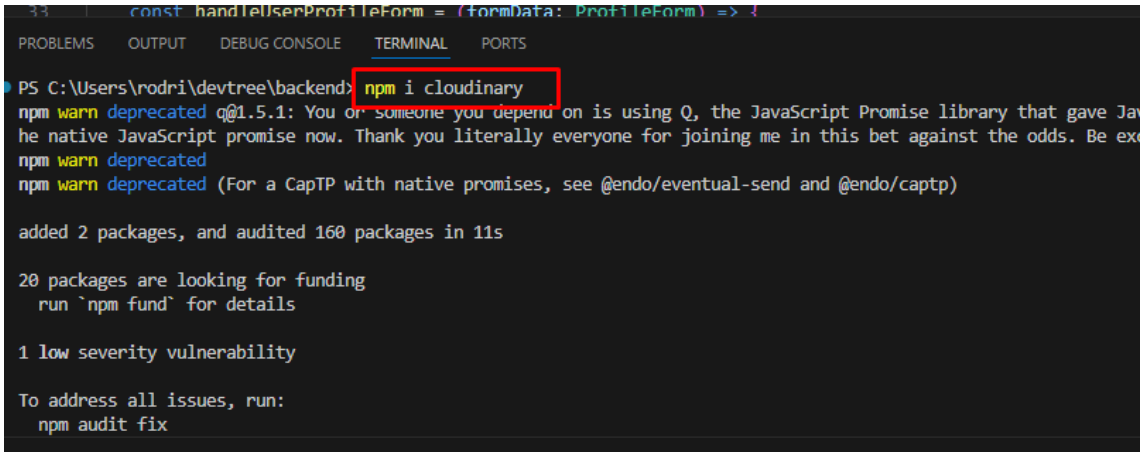
Primeros pasos y crear cuenta en Cloudinary

Muchos proyectos hoy en día cuentan con subida de archivos. Los archivos no deben almacenarse en la base de datos, únicamente su ubicación en el servidor. Si tu proyecto tiene un gran volumen de subida de archivos lo mejor será hospedar las imágenes en un servidor separado o contratar un servicio como Amazon S3, Firebase o Cloudinary. Express no cuenta con subida de archivos, pero se puede hacer por medio de una dependencia como Multer o Formidable.

https://cloudinary.com/

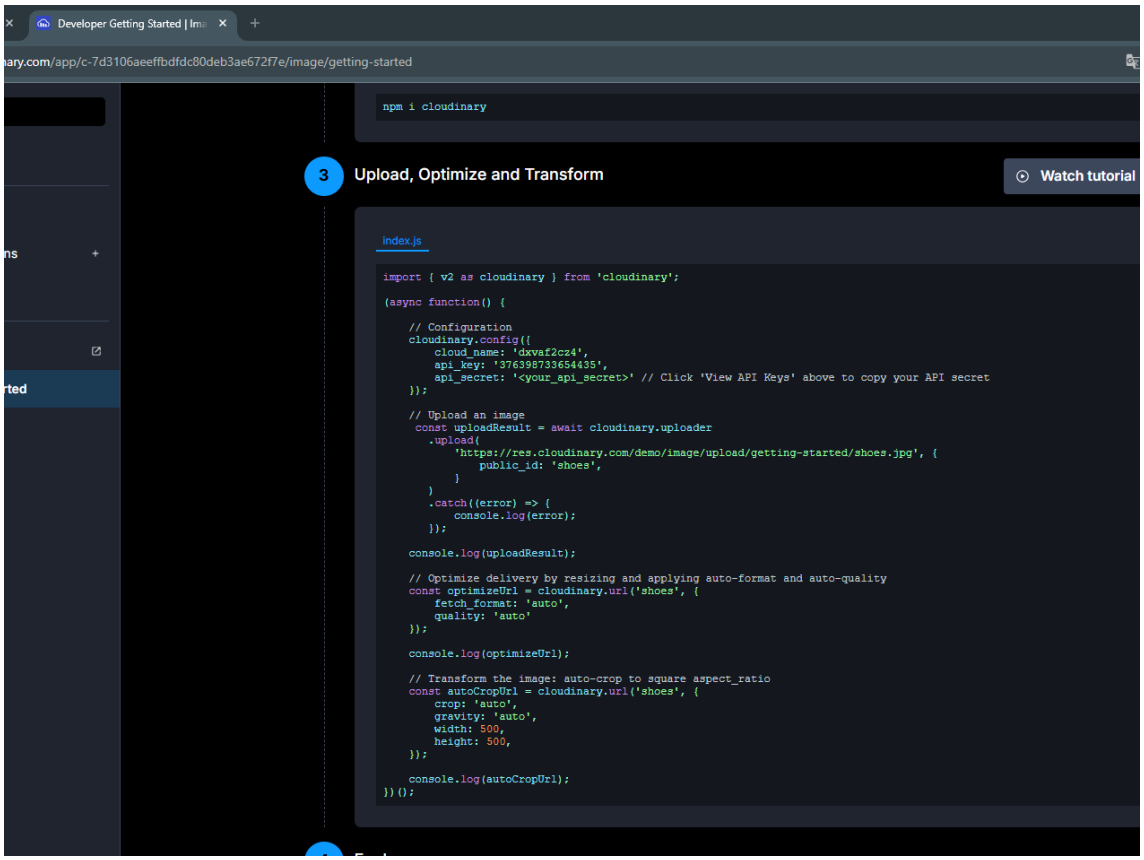
Registrarse.





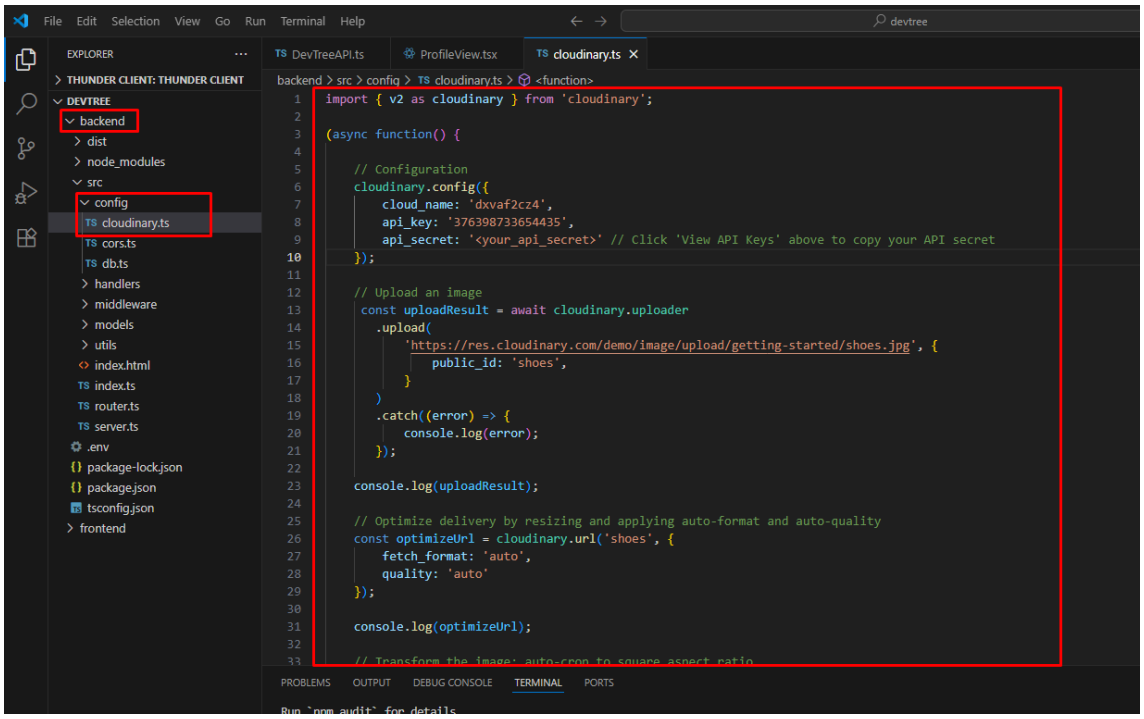
Instalamos “formidable”, permitirá que express pueda soportar la subida de archivos, y cloudinary es donde se van a almacenar.

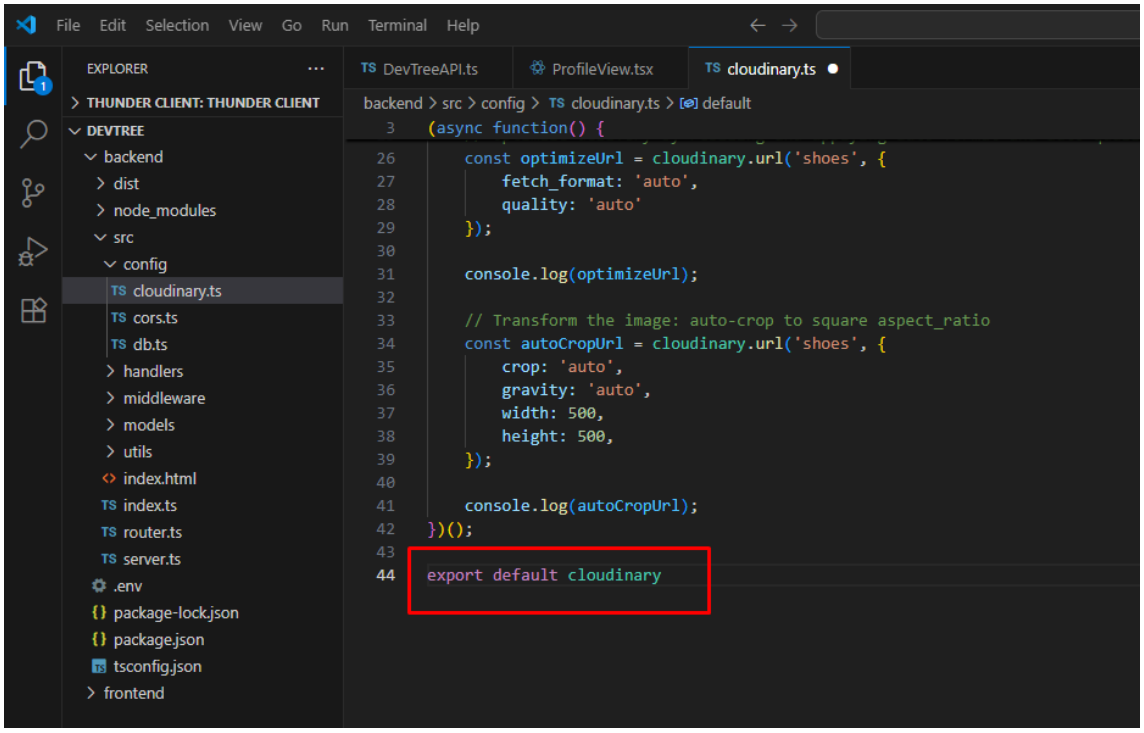
```
PS C:\Users\rodri\devtree\backend> npm i formidable

added 5 packages, and audited 165 packages in 5s

22 packages are looking for funding
  run `npm fund` for details

1 low severity vulnerability

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
PS C:\Users\rodri\devtree\backend>
```



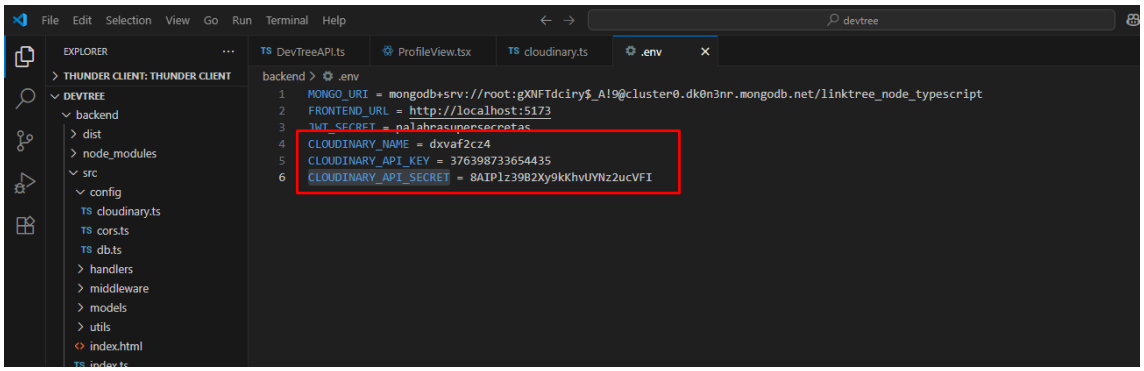
Agregando las credenciales de Cloudinary en el backend

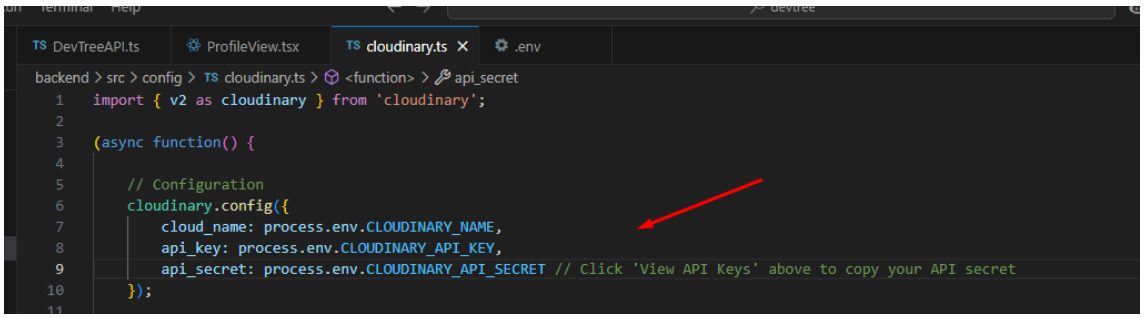
Crear cloudinary.ts y Copiar:





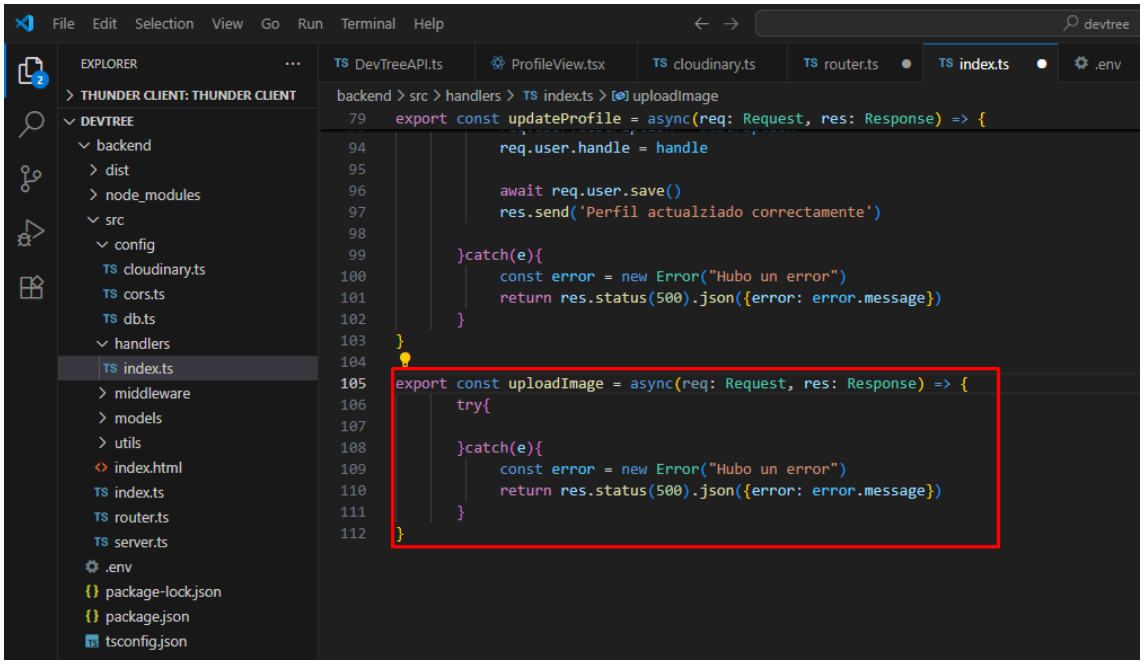
Ahora, ahí tenemos información sensible, entonces tenemos que ocultarlo en variables de entorno.

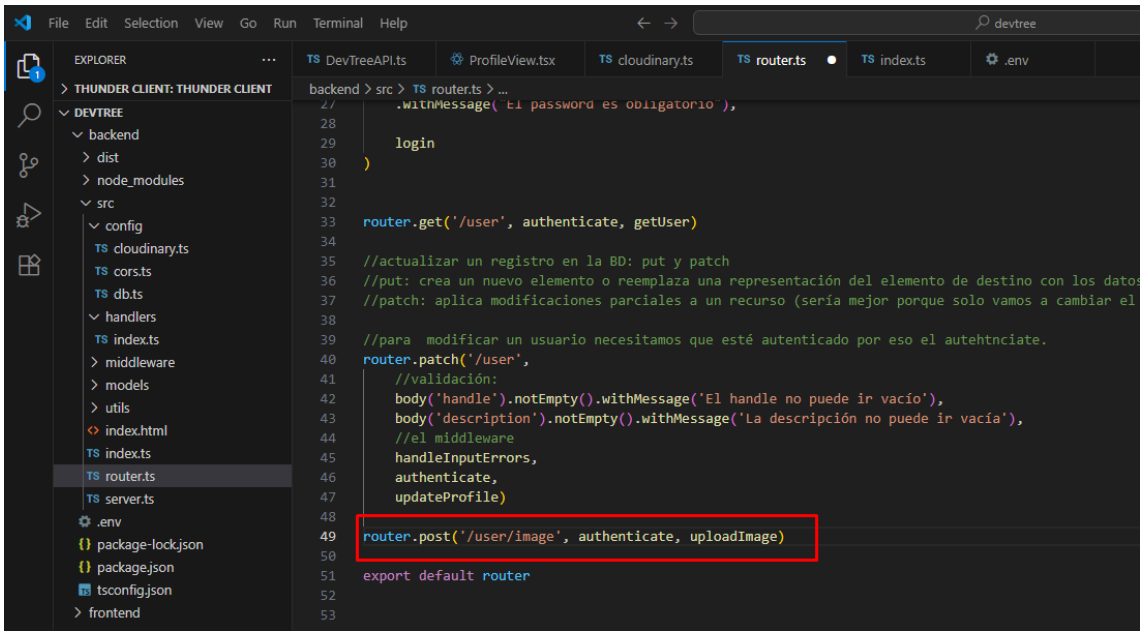




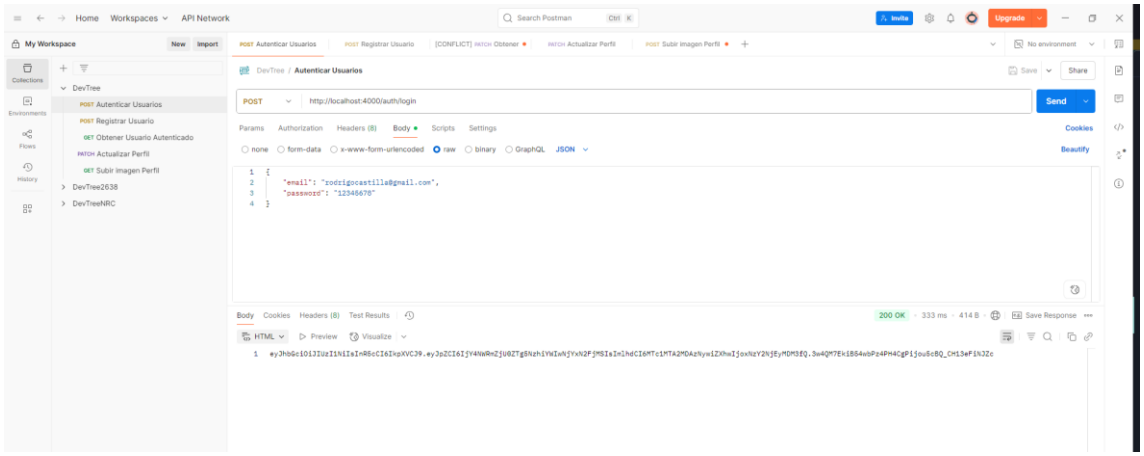
Leyendo imágenes con Formidable

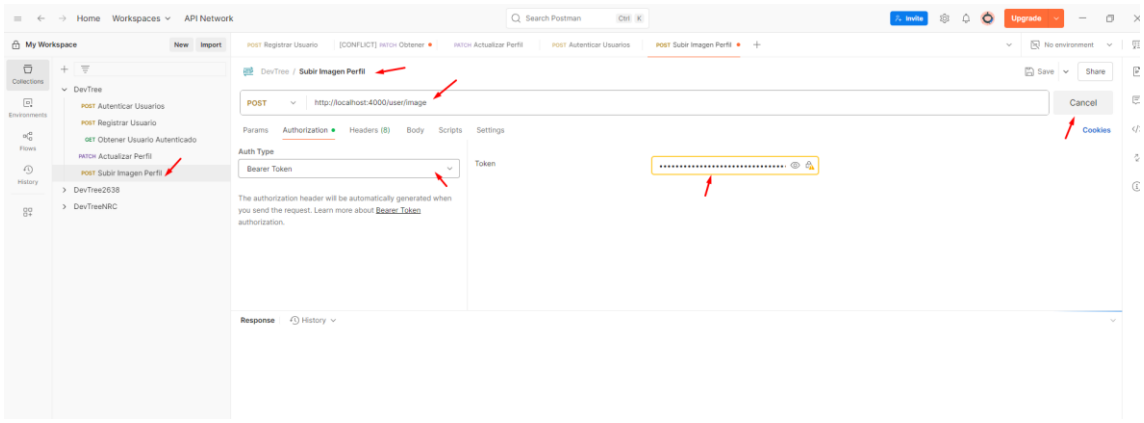
Creando ruta para enviar la imagen, cuando subamos la imagen enviamos la petición hacia la URL





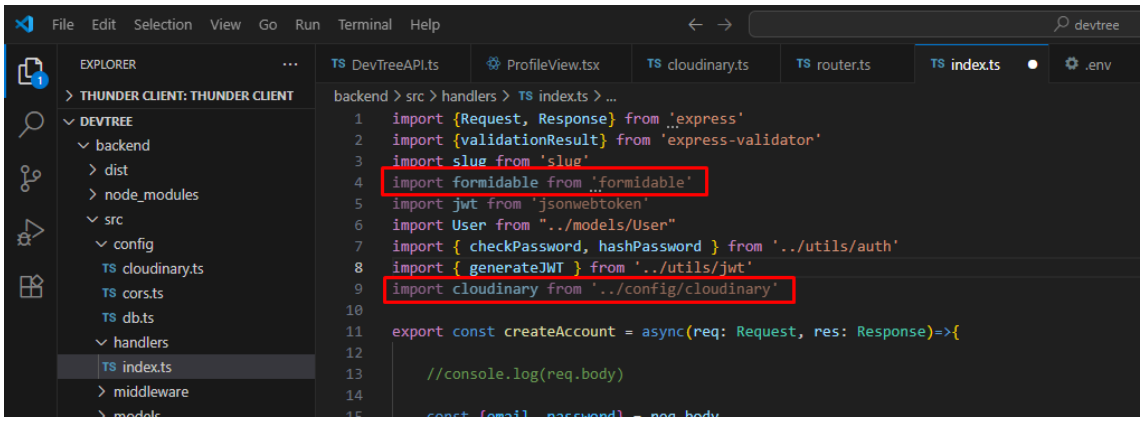
Autenticar usuarios para tener un token nuevo.





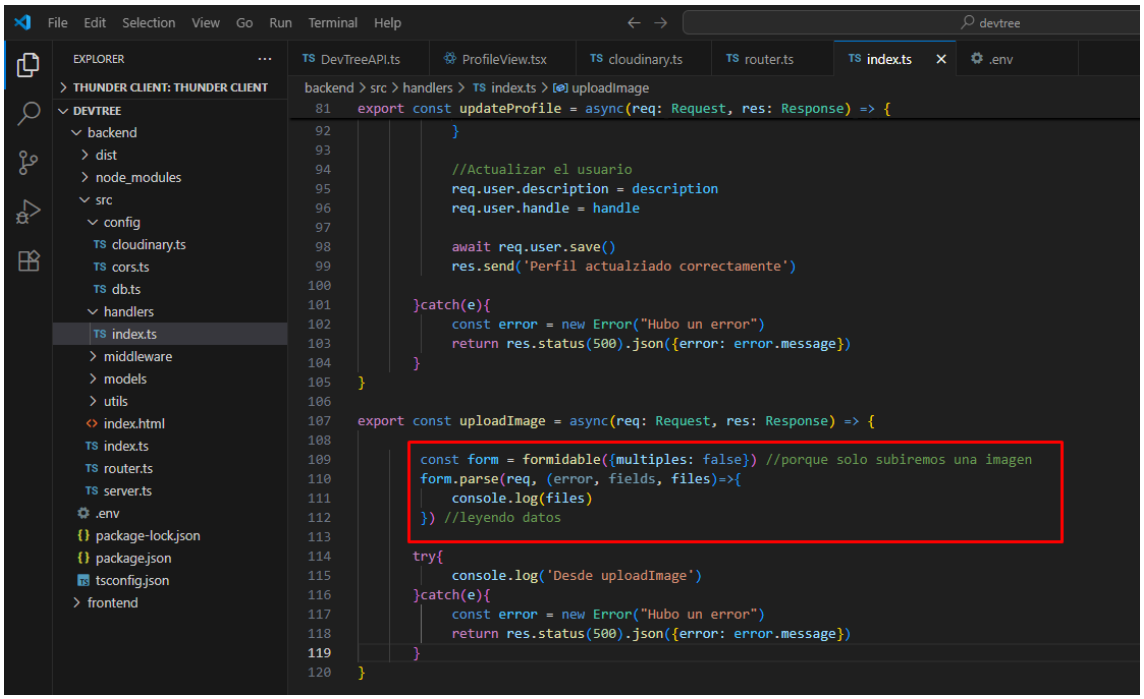
```
PS C:\Users\rodri\devtree\backend> npm run dev:api

> devtree@1.0.0 dev:api
> nodemon src/index.ts --api

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto  4000
Desde validation.ts
Desde uploadImage
```

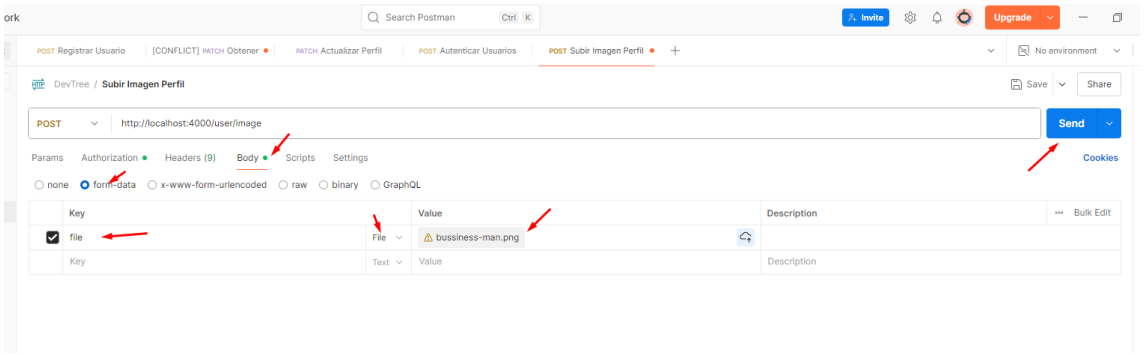
Ahora, express no soporta subida de archivos, es un framework que poco a poco vamos agregándole lo que requerimos. Usaremos formidable.

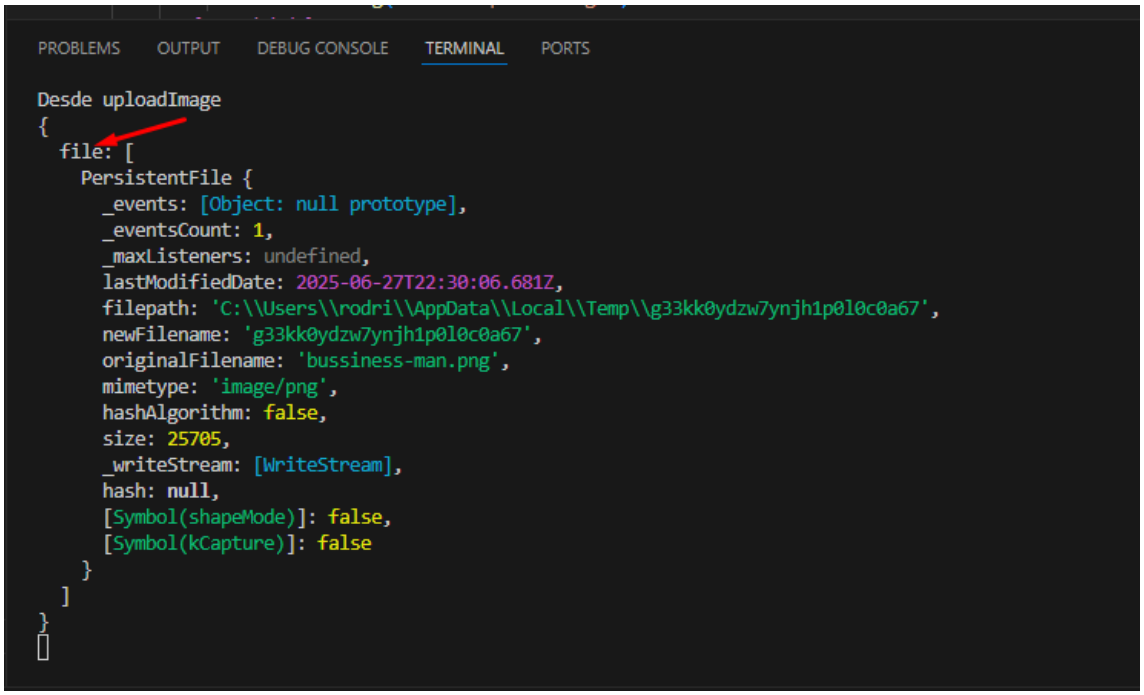


Leer datos del formulario:

```
Desde uploadImage
PS C:\Users\rodri\devtree\backend> npm i --save-dev @types/formidable

added 1 package, and audited 166 packages in 11s

22 packages are looking for funding
  run `npm fund` for details

1 low severity vulnerability

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
PS C:\Users\rodri\devtree\backend>
```


File Edit Selection View Go Run Terminal Help
devtree
EXPLORER
THUNDER CLIENT: THUNDER CLIENT
DEVTREE
backend
dist
node_modules
src
config
cloudinary.ts
cors.ts
db.ts
handlers
index.ts
middleware
models
utils
index.html
index.ts
router.ts
server.ts
.env
package-lock.json
package.json
tsconfig.json
frontend
DevTreeAPI.ts
ProfileView.tsx
cloudinary.ts
router.ts
index.ts
.env
backend > src > handlers > index.ts > uploadImage
export const updateProfile = async(req: Request, res: Response) => {
}
//Actualizar el usuario
req.user.description = description
req.user.handle = handle
await req.user.save()
res.send('Perfil actualziado correctamente')
}catch(e){
const error = new Error("Hubo un error")
return res.status(500).json({error: error.message})
}
}
export const uploadImage = async(req: Request, res: Response) => {
const form = formidable({multiples: false}) //porque solo subiremos una imagen
form.parse(req, (error, fields, files)=>{
console.log(files)
}) //leyendo datos
try{
console.log('Desde uploadImage')
}catch(e){
const error = new Error("Hubo un error")
return res.status(500).json({error: error.message})
}
}


Search Postman
Invite
Upgrade
POST Registrar Usuario
[CONFLICT] PATCH Obtener
PATCH Actualizar Perfil
POST Autenticar Usuarios
POST Subir Imagen Perfil
No environment
DevTree / Subir Imagen Perfil
Save
Share
POST
http://localhost:4000/user/image
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Description
Bulk Edit
file
File
bussiness-man.png
Key
Text
Value
Description


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Desde uploadImage
{
file: [
PersistentFile {
_events: [Object: null prototype],
_eventsCount: 1,
_maxListeners: undefined,
lastModifiedDate: 2025-06-27T22:30:06.681Z,
filepath: 'C:\\Users\\rodri\\AppData\\Local\\Temp\\g33kk0ydzw7ynjh1p0l0c0a67',
newFilename: 'g33kk0ydzw7ynjh1p0l0c0a67',
originalFilename: 'bussiness-man.png',
mimetype: 'image/png',
hashAlgorithm: false,
size: 25705,
_writeStream: [WriteStream],
hash: null,
[Symbol(shapeMode)]: false,
[Symbol(kCapture)]: false
}
]
}

```
export const uploadImage = async(req: Request, res: Response) => {

        const form = formidable({multiples: false}) //porque solo subiremos una imagen
        form.parse(req, (error, fields, files)=>{
            console.log(files.file)  //para que no tenga que acceder al objeto
        }) //leyendo datos

        try{
            console.log('Desde uploadImage')
        }catch(e){
            const error = new Error("Hubo un error")
            return res.status(500).json({error: error.message})
```

Tenemos mucha información en consola: la ruta, un nombre nuevo, el nombre original. Cancelamos en postman.

Lo que tenemos que subir a cloudinary es el filepath porque eso está en memoria en nuestro servidor, que tiene que ser procesado y debe subirse.

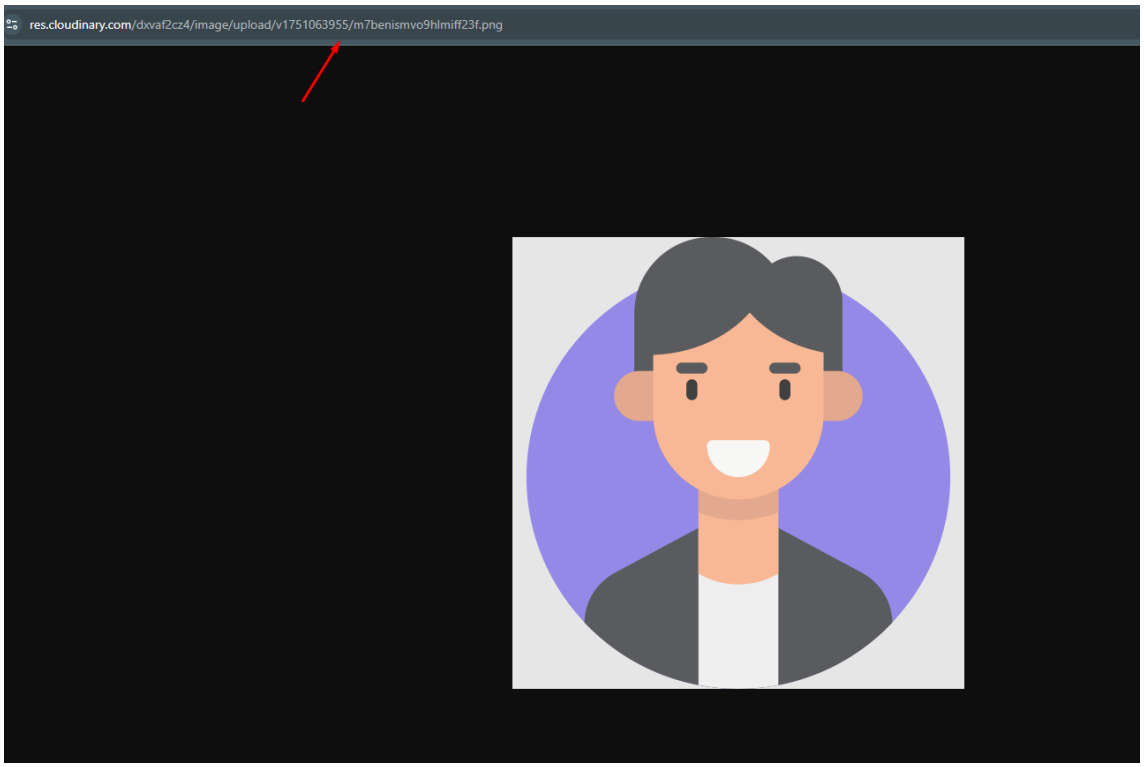
Subiendo imágenes a Cloudinary

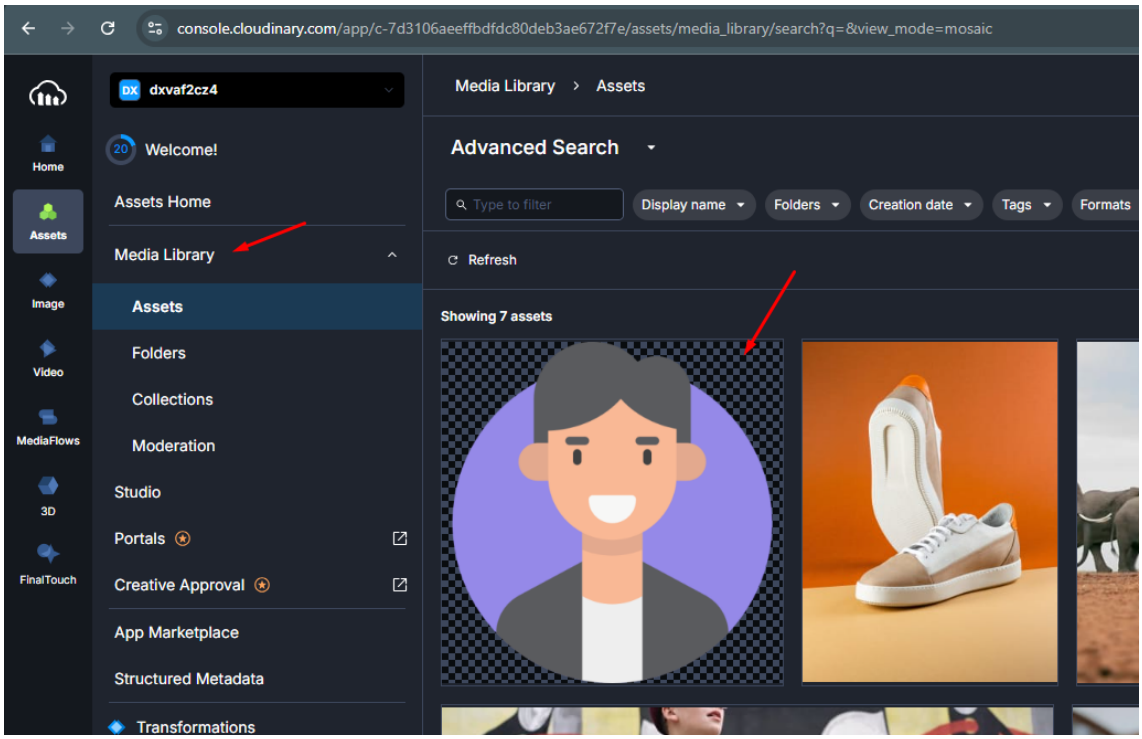
```
export const uploadImage = async(req: Request, res: Response) => {

        const form = formidable({multiples: false}) //porque solo subiremos una imagen
        form.parse(req, (error, fields, files)=>{
            //console.log()  //para que no tenga que acceder al objeto
            cloudinary.uploader.upload(files.file[0].filepath, {}, async function(error, result){
                //va a ser async porque va a interactuar con la api o cloudinary
                console.log(error)
                console.log(result)
            } )
        })

        try{
            console.log('Desde uploadImage')
        }catch(e){
            const error = new Error("Hubo un error")
```

Reiniciamos servidor, enviamos nuevamente desde postman.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

{
  asset_id: 'ac232ff1b0193921d8cf164c90c996c8',
  public_id: 'm7benismvo9hlmiff23f',
  version: 1751063955,
  version_id: '1cf3c72758afea67fc91f0e017c8574b',
  signature: 'ad72306937cee305f4ba5849765a4797125de880',
  width: 512,
  height: 512,
  format: 'png',
  resource_type: 'image',
  created_at: '2025-06-27T22:39:15Z',
  tags: [],
  bytes: 25705,
  type: 'upload',
  etag: '7b492df6877444ddd6caddc8a0f72e23',
  placeholder: false,
  url: 'http://res.cloudinary.com/dxvaf2cz4/image/upload/v1751063955/m7benismvo9hlmiff23f.png',
  secure_url: 'https://res.cloudinary.com/dxvaf2cz4/image/upload/v1751063955/m7benismvo9hlmiff23f.png',
  asset_folder: '',
  display_name: 'm7benismvo9hlmiff23f',
```

Toda esta información viene desde cloudinary.





Almacenaremos la URL en la base de datos. Almacenaremos secure_url.

```
export const uploadImage = async(req: Request, res: Response) => {

        const form = formidable({multiples: false}) //porque solo subiremos una imagen
        form.parse(req, (error, fields, files)=>{
            //console.log()  //para que no tenga que acceder al objeto
            cloudinary.uploader.upload(files.file[0].filepath, {}, async function(error, result){
                //va a ser async porque va a interactuar con la api o cloudinary
                //console.log(error)
                //console.log(result)
                if(error){
                    const error = new Error("Hubo un error al subir la imagen")
                    return res.status(500).json({error: error.message})
                }
                if(result){
                    console.log(result.secure_url)
                }
            } )
        })
```

Enviamos nuevamente desde postman, y en consola ya tenemos la secure_url

```
https://res.cloudinary.com/dxvaf2cz4/image/upload/f_auto,q_auto/shoes?_a=BAMAK+Xy0
https://res.cloudinary.com/dxvaf2cz4/image/upload/c_auto,g_auto,h_500,w_500/shoes?_a=BAMAK+Xy0
Desde uploadImage
https://res.cloudinary.com/dxvaf2cz4/image/upload/v1751064231/ft3jadfvemfwlmvtekmi.png
```

Esta es la que vamos a asociar a la bd con el usuario autenticado, pero aun no tenemos en el modelo de usuario donde almacenar la imagen. Pero antes.

Public_id: para generar un nombre único

Eso podemos hacerlo con una librería: UniqueId

```
PS C:\Users\rodri\devtree> cd backend
PS C:\Users\rodri\devtree\backend> npm i uuid
```

```
ckend > src > handlers > TS index.ts > createAccount
1   import {Request, Response} from 'express'
2   import {validationResult} from 'express-validator'
3   import slug from 'slug'
4   import formidable from 'formidable'
5   import {v4 as uuid} from 'uuid'
6   import jwt from 'jsonwebtoken'
7   import User from "../models/User"
8   import { checkPassword, hashPassword } from '../utils/auth'
9   import { generateJWT } from '../utils/jwt'
```

```
export const uploadImage = async(req: Request, res: Response) => {

        const form = formidable({multiples: false}) //porque solo subiremos una imagen
        form.parse(req, (error, fields, files)=>{
            //console.log()  //para que no tenga que acceder al objeto
            cloudinary.uploader.upload(files.file[0].filepath, {public_id: uuid()}, async function(error, result){
                //va a ser async porque va a interactuar con la api o cloudinary
                //console.log(error)
                //console.log(result)
```

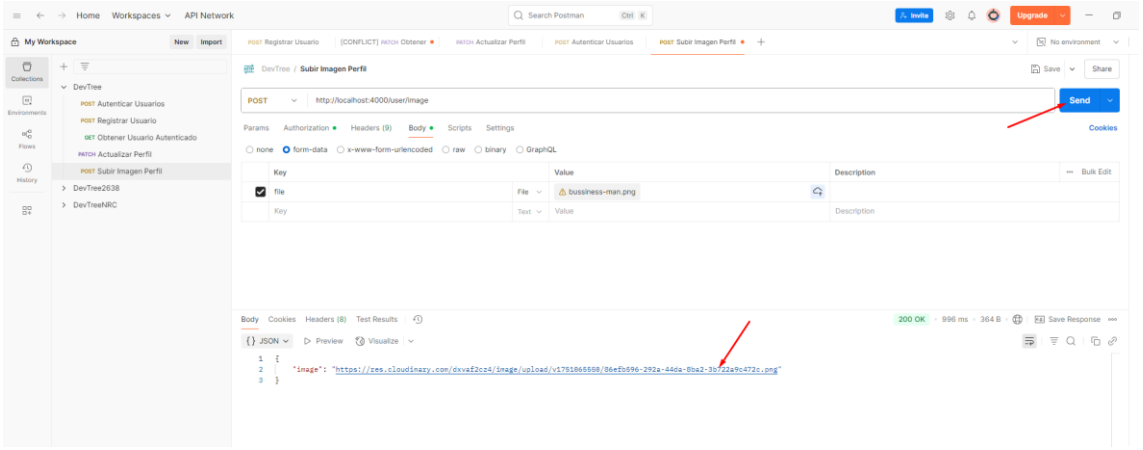
Asignando la imagen al usuario autenticado

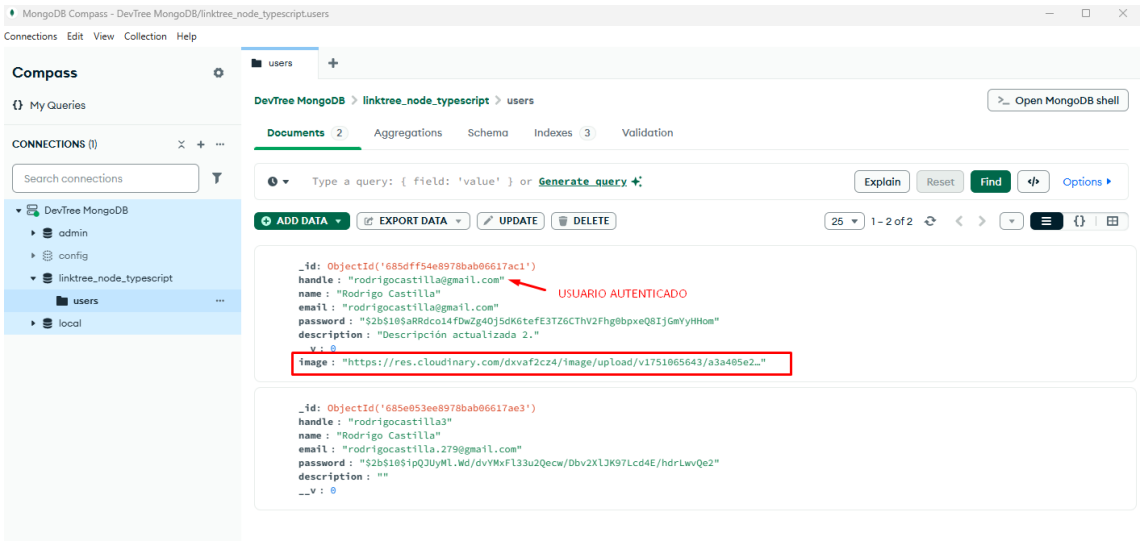
```
backend > src > models > TS User.ts > userSchema
1   import mongoose, {Schema, Document} from 'mongoose'
2
3   //Al agregarle Document estamos heredando todo lo que tiene Document(atributos) de Mongoose
4   export interface IUser extends Document {
5       handle: string
6       name: string
7       email: string
8       password: string
9       description: string
10      image: string
11
12  }
```

```
        default: ''
    },
    image: {
        type: String,
        default: ''
    }

})
```

Index.ts

```
export const uploadImage = async(req: Request, res: Response) => {

    const form = formidable({multiples: false}) //porque solo subiremos una imagen
    form.parse(req, (error, fields, files)=>{
        //console.log()  //para que no tenga que acceder al objeto
        cloudinary.uploader.upload(files.file[0].filepath, {public_id: uuid()}, async function(error, result){
            //va a ser async porque va a interactuar con la api o cloudinary
            //console.log(error)
            //console.log(result)
            if(error){
                const error = new Error("Hubo un error al subir la imagen")
                return res.status(500).json({error: error.message})
            }
            if(result){
                req.user.iamge = result.secure_url
                await req.user.save()
                res.json({image: result.secure_url})
            }
        } )
    })
```

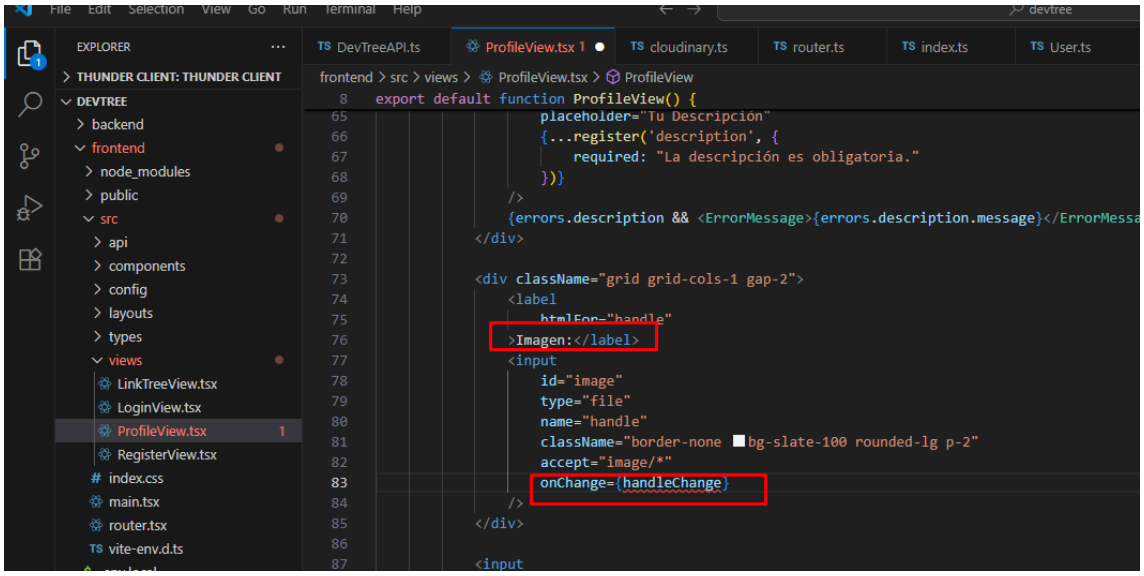


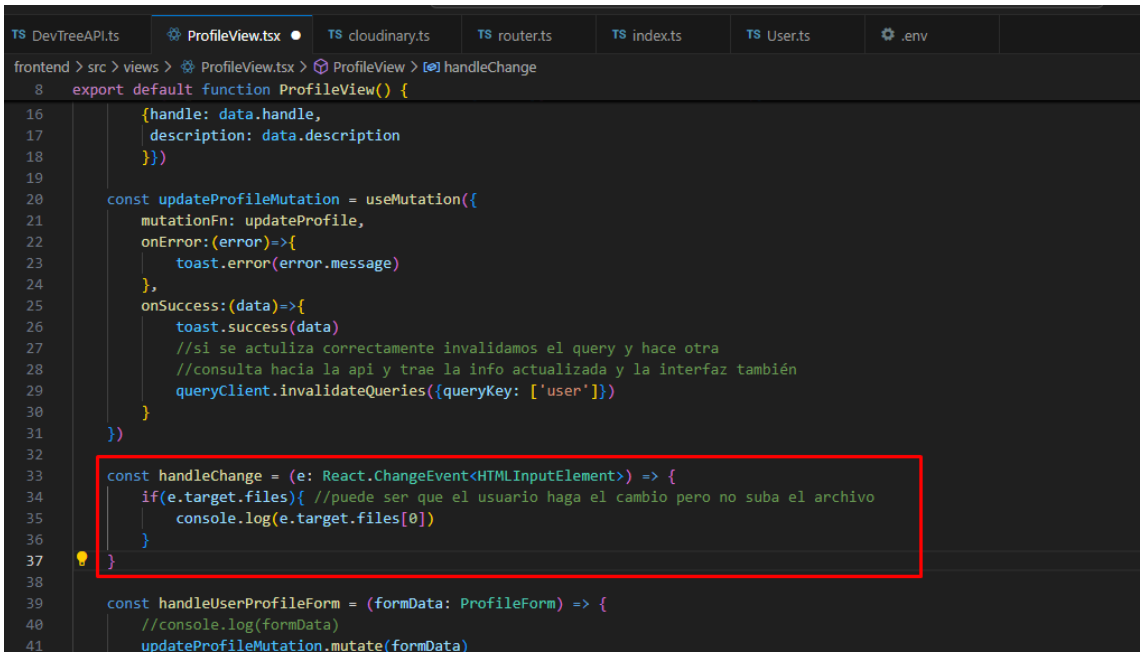


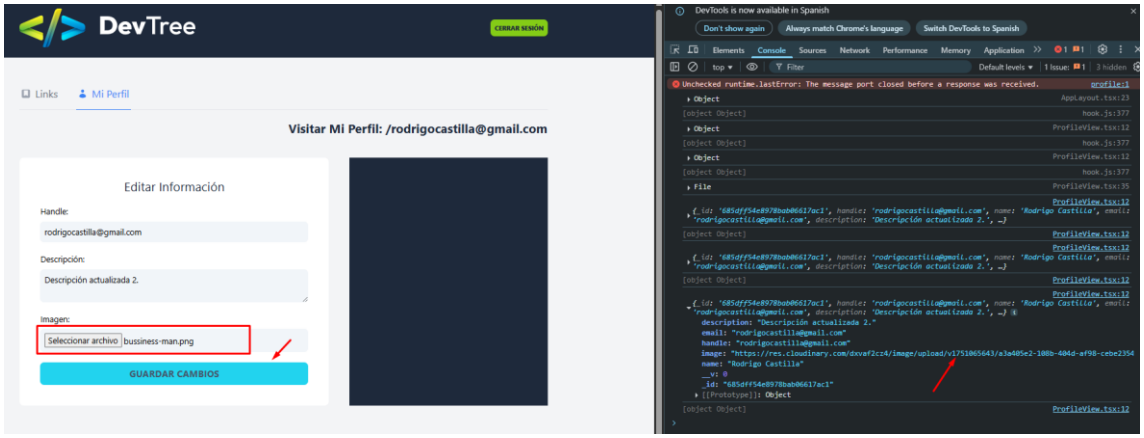
Entonces ya la podemos renderizar.

Con eso ya tenemos listo en el backend, y ahora veamos como hacerlo en el frontend.

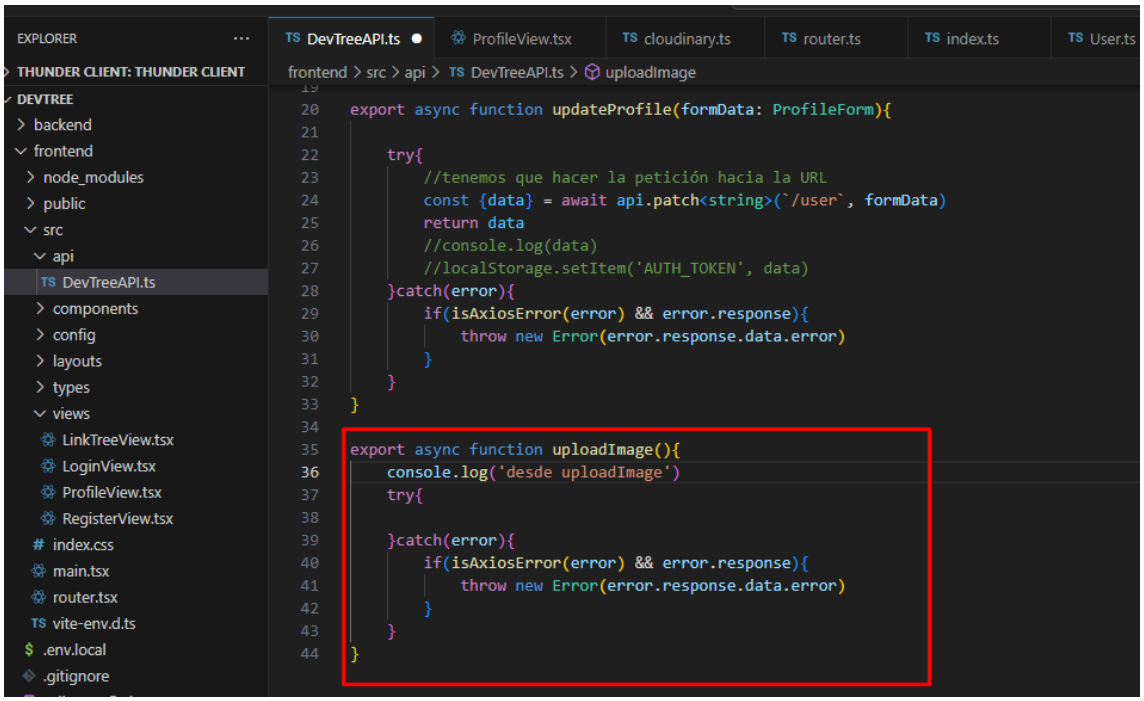
Leyendo el archivo que se sube en React



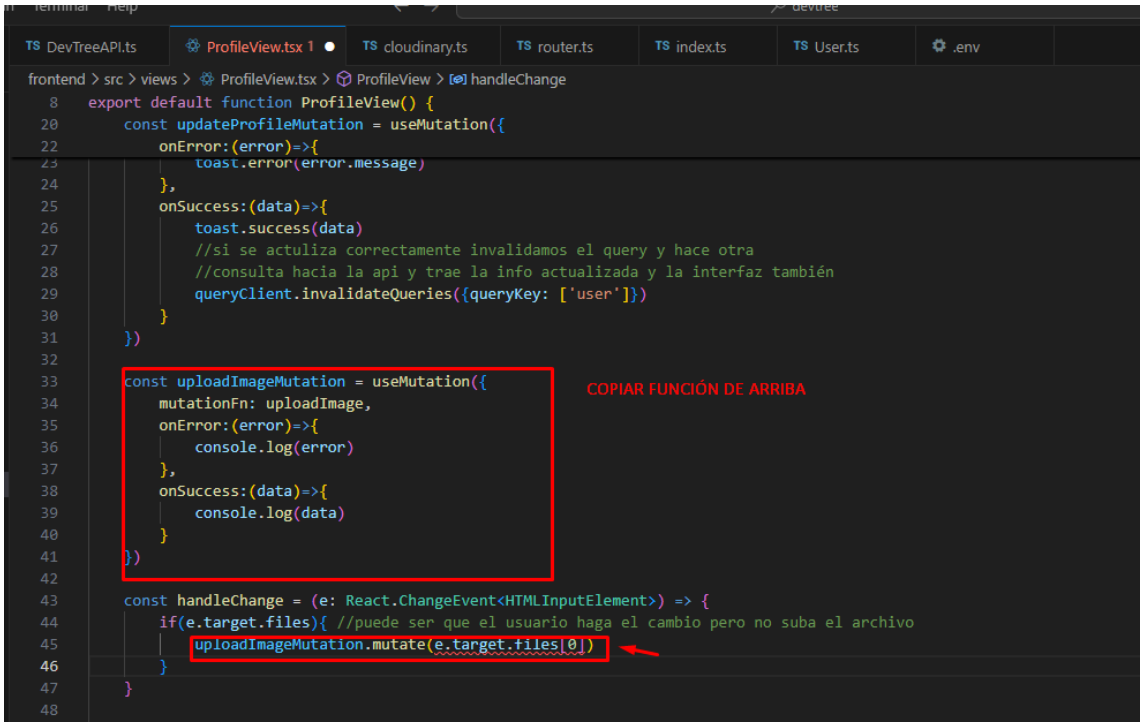




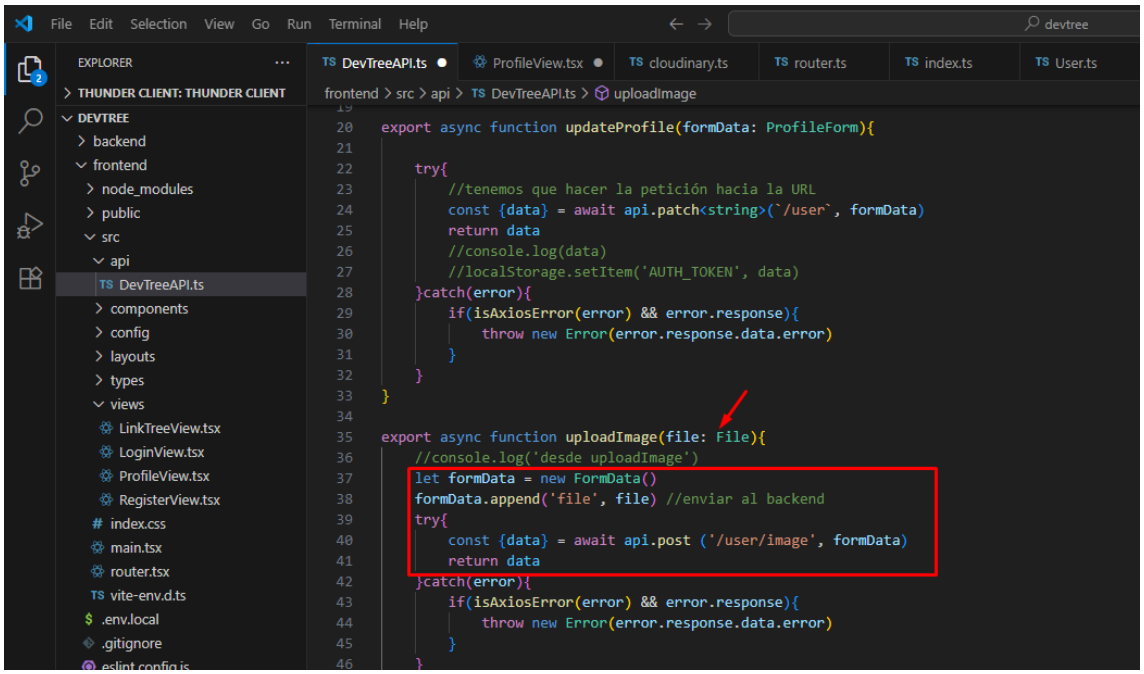
Enviando la imagen con Axios y ReactQuery



Ahora hay que crear la mutación.

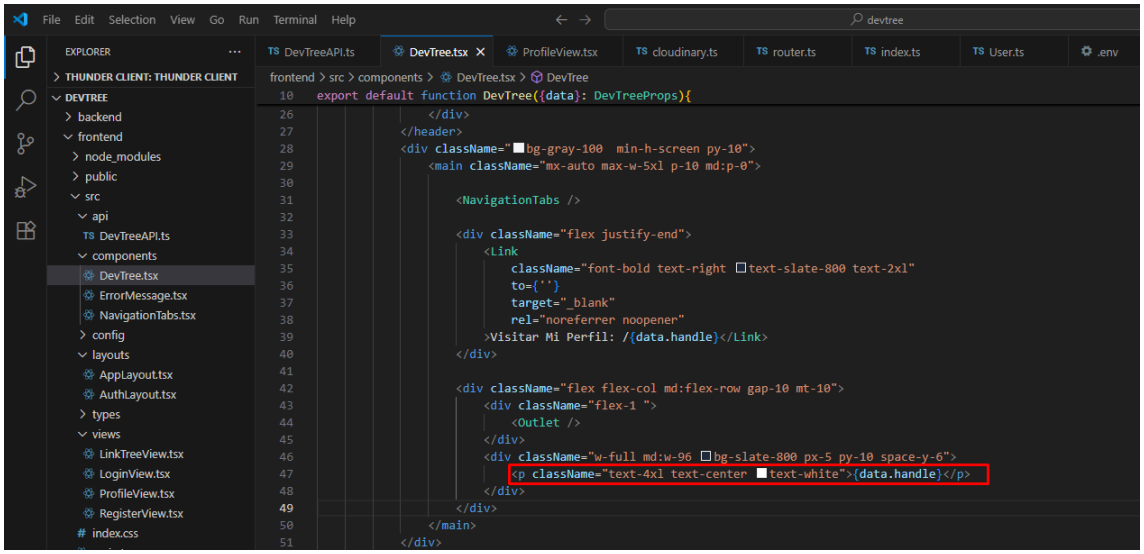


Sale rojo, entonces, tenemos que pasar el archivo completo en DevTreeAPI, no es únicamente un string.



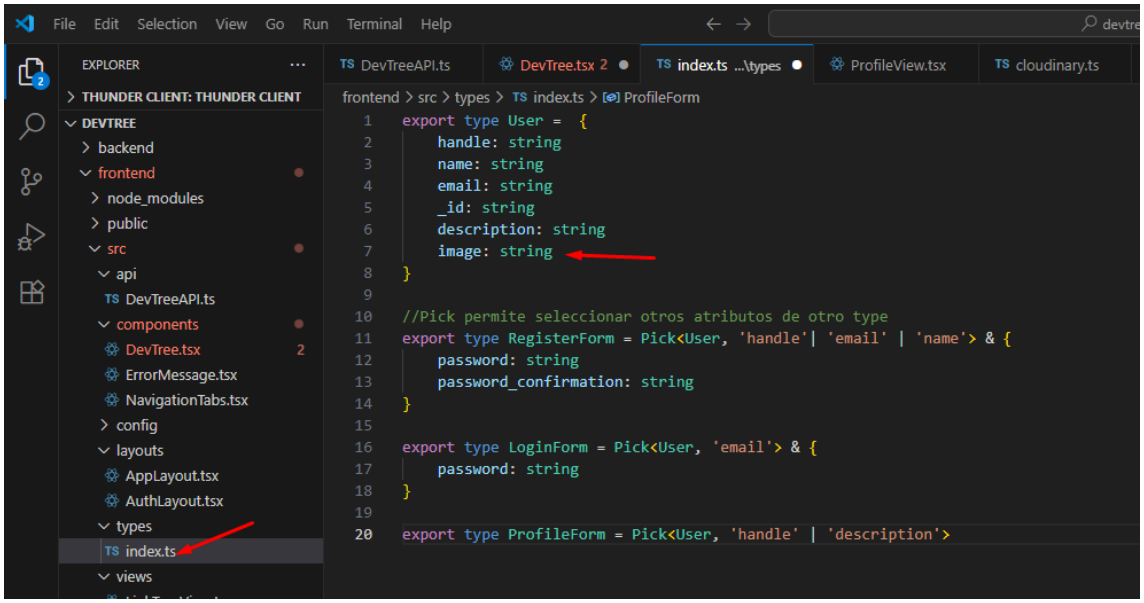
```ts
export async function updateProfile(formData: ProfileForm){

    try{
        //tenemos que hacer la petición hacia la URL
        const {data} = await api.patch<string>(`/user`, formData)
        return data
        //console.log(data)
        //localStorage.setItem('AUTH_TOKEN', data)
    }catch(error){
        if(isAxiosError(error) && error.response){
            throw new Error(error.response.data.error)
        }
    }
}

export async function uploadImage(file: File){
    //console.log('desde uploadImage')
    let formData = new FormData()
    formData.append('file', file) //enviar al backend
    try{
        const {data} = await api.post ('/user/image', formData)
        return data
    }catch(error){
        if(isAxiosError(error) && error.response){
            throw new Error(error.response.data.error)
        }
    }
```

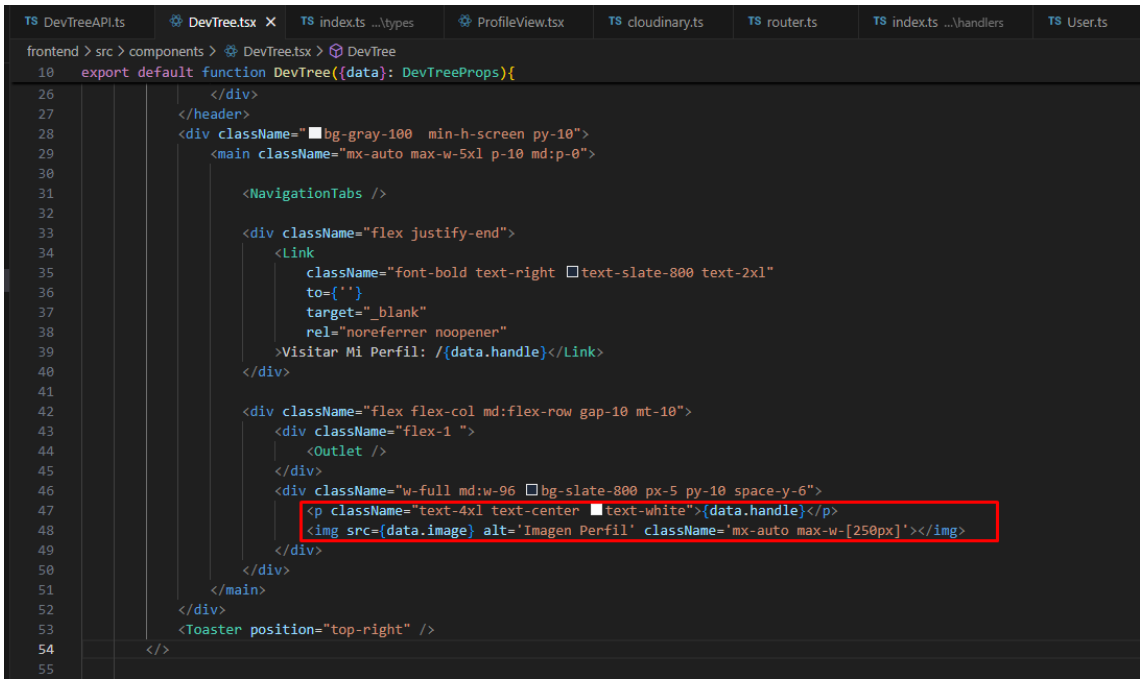
Mostrando la información de Perfil e Imagen



```tsx
            </div>
        </header>
        <div className="bg-gray-100  min-h-screen py-10">
            <main className="mx-auto max-w-5xl p-10 md:p-0">

                <NavigationTabs />

                <div className="flex justify-end">
                    <Link
                        className="font-bold text-right text-slate-800 text-2xl"
                        to={''}
                        target="_blank"
                        rel="noreferrer noopener"
                    >Visitar Mi Perfil: /{data.handle}</Link>
                </div>

                <div className="flex flex-col md:flex-row gap-10 mt-10">
                    <div className="flex-1 ">
                        <Outlet />
                    </div>
                    <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
                        <p className="text-4xl text-center text-white">{data.handle}</p>
                    </div>
                </div>
            </main>
        </div>
```

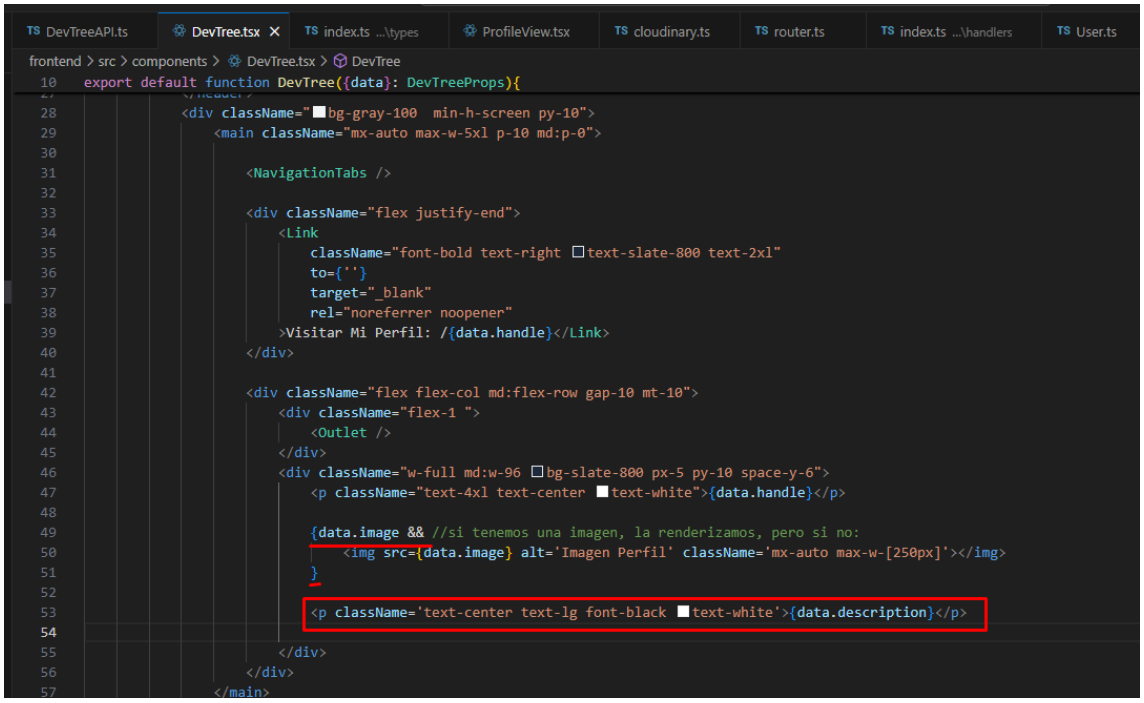

DevTree
CERRAR SESIÓN
Links
Mi Perfil
Visitar Mi Perfil: /rodrigocastilla
Editar Información
Handle:
rodrigocastilla
Descripción:
Descripción actualizada 2.
Imagen:
rodrigocastilla


File Edit Selection View Go Run Terminal Help
EXPLORER
THUNDER CLIENT: THUNDER CLIENT
DEVTREE
backend
frontend
node_modules
public
src
api
DevTreeAPI.ts
components
DevTree.tsx
ErrorMessage.tsx
NavigationTabs.tsx
config
layouts
AppLayout.tsx
AuthLayout.tsx
types
index.ts
views
DevTreeAPI.ts
DevTree.tsx 2
index.ts ...\types
ProfileView.tsx
cloudinary.ts
frontend > src > types > index.ts > ProfileForm
export type User = {
handle: string
name: string
email: string
_id: string
description: string
image: string
}
//Pick permite seleccionar otros atributos de otro type
export type RegisterForm = Pick<User, 'handle'| 'email' | 'name'> & {
password: string
password_confirmation: string
}
export type LoginForm = Pick<User, 'email'> & {
password: string
}
export type ProfileForm = Pick<User, 'handle' | 'description'>

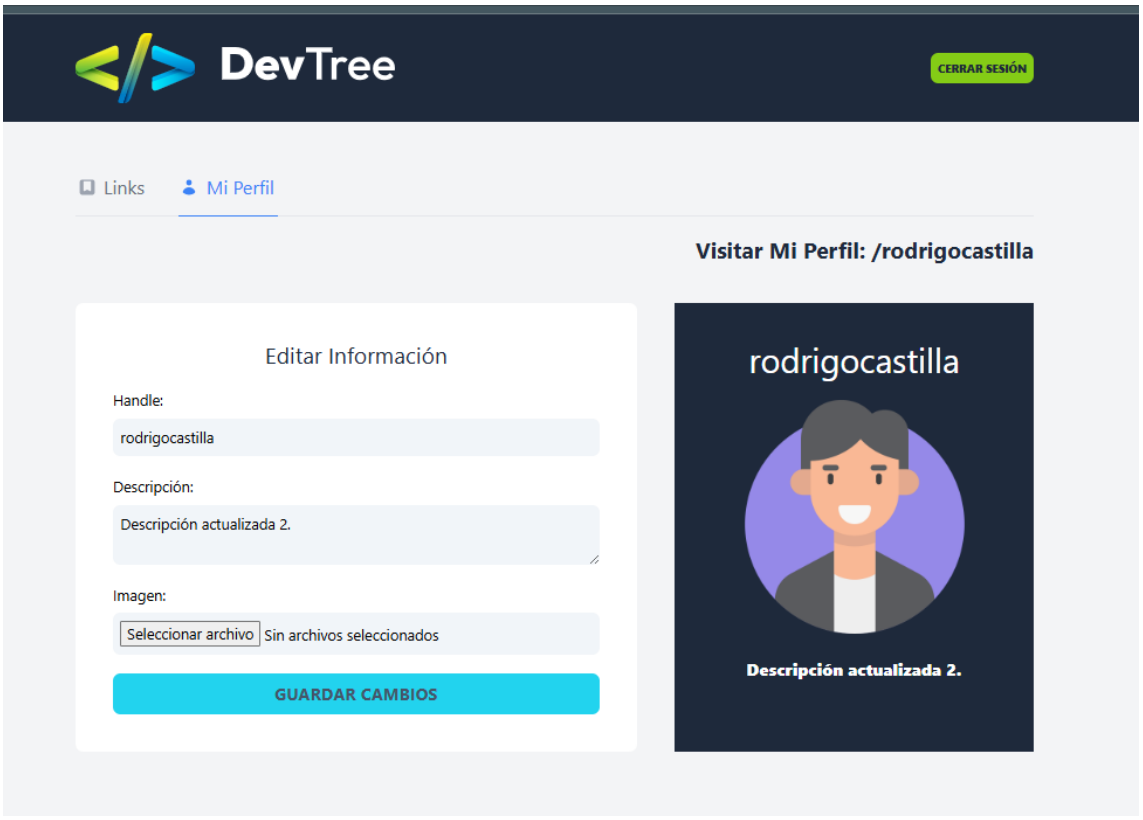


```
export default function DevTree({data}: DevTreeProps){
                </div>
            </header>
            <div className="bg-gray-100  min-h-screen py-10">
                <main className="mx-auto max-w-5xl p-10 md:p-0">

                    <NavigationTabs />

                    <div className="flex justify-end">
                        <Link
                            className="font-bold text-right text-slate-800 text-2xl"
                            to={''}
                            target="_blank"
                            rel="noreferrer noopener"
                        >Visitar Mi Perfil: /{data.handle}</Link>
                    </div>

                    <div className="flex flex-col md:flex-row gap-10 mt-10">
                        <div className="flex-1 ">
                            <Outlet />
                        </div>
                        <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
                            <p className="text-4xl text-center text-white">{data.handle}</p>
                            <img src={data.image} alt='Imagen Perfil' className='mx-auto max-w-[250px]'></img>
                        </div>
                    </div>
                </main>
            </div>
            <Toaster position="top-right" />
        </>
```

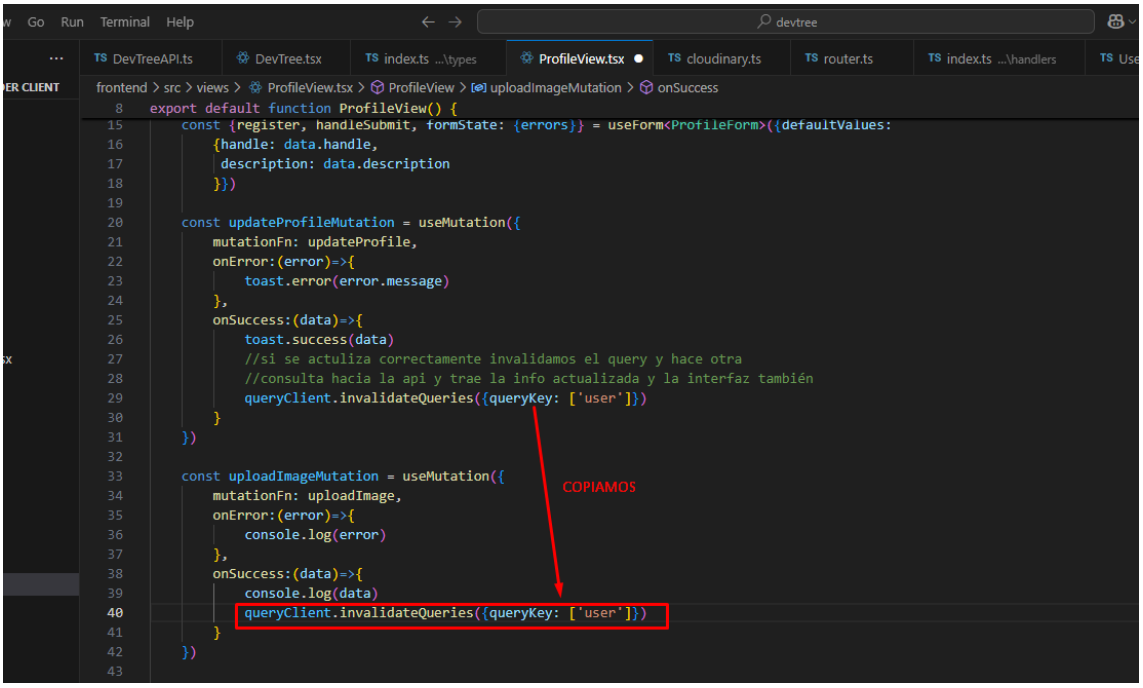
Para usuarios que no tengan foto, les saldrá un enlace roto. Lo arreglamos:



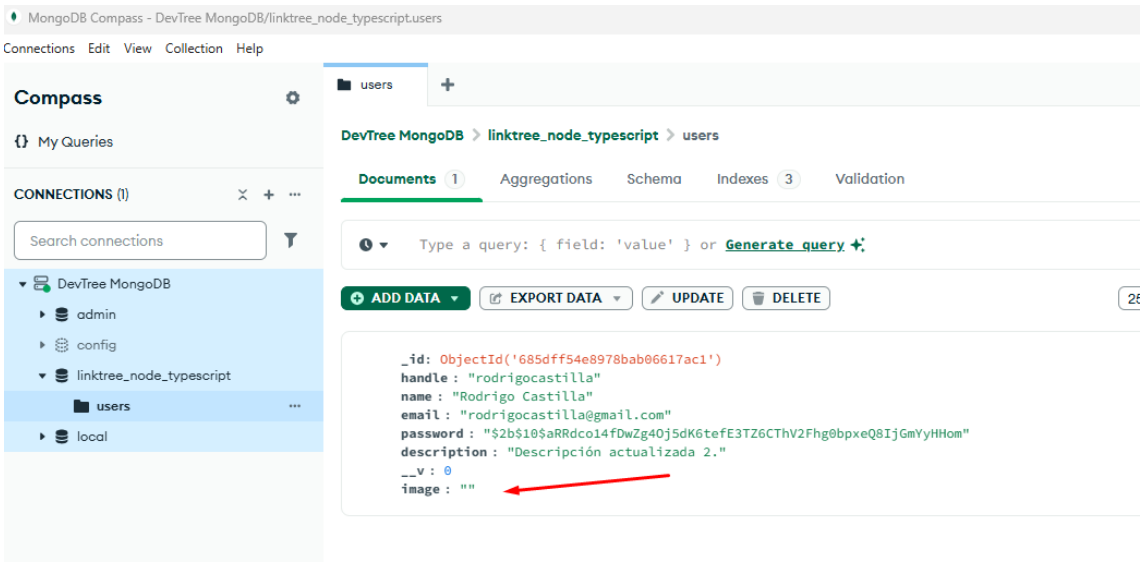
```
export default function DevTree({data}: DevTreeProps){
            <div className="bg-gray-100  min-h-screen py-10">
                <main className="mx-auto max-w-5xl p-10 md:p-0">

                    <NavigationTabs />

                    <div className="flex justify-end">
                        <Link
                            className="font-bold text-right text-slate-800 text-2xl"
                            to={''}
                            target="_blank"
                            rel="noreferrer noopener"
                        >Visitar Mi Perfil: /{data.handle}</Link>
                    </div>

                    <div className="flex flex-col md:flex-row gap-10 mt-10">
                        <div className="flex-1 ">
                            <Outlet />
                        </div>
                        <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
                            <p className="text-4xl text-center text-white">{data.handle}</p>

                            {data.image && //si tenemos una imagen, la renderizamos, pero si no:
                                <img src={data.image} alt='Imagen Perfil' className='mx-auto max-w-[250px]'></img>
                            }

                            <p className='text-center text-lg font-black text-white'>{data.description}</p>

                        </div>
                    </div>
                </main>
```

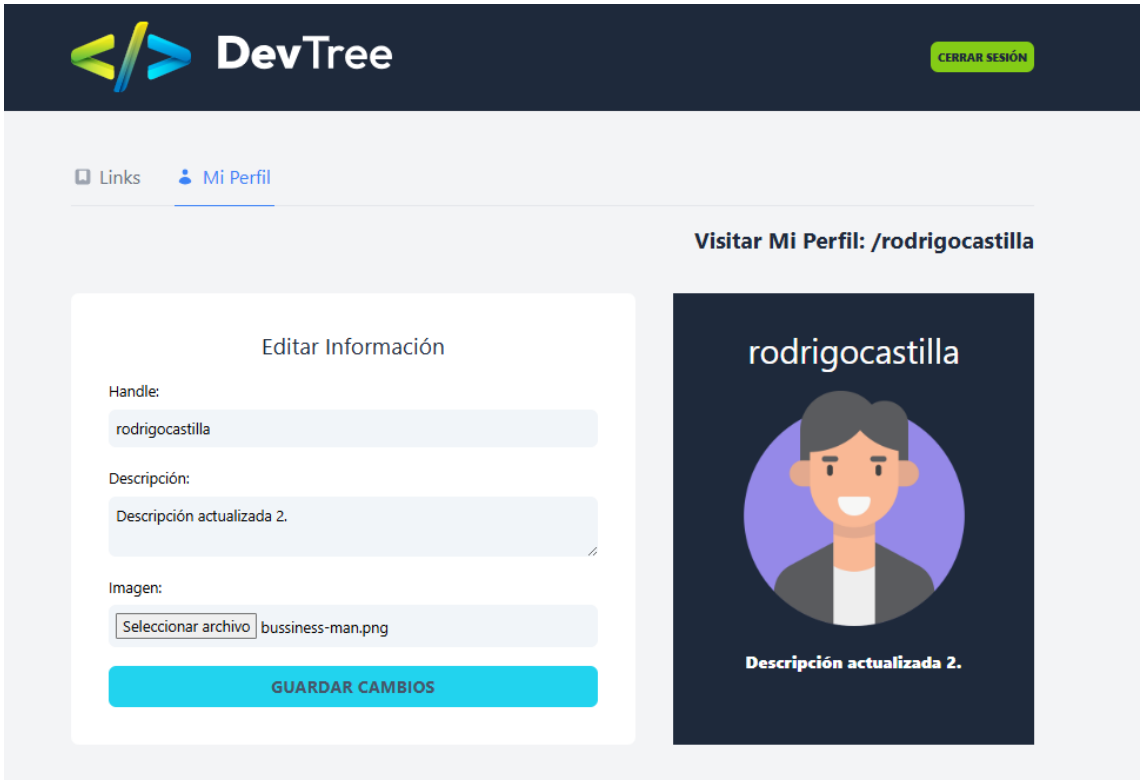


Entonces vemos que al actualizar la descripción o el handle se actualiza, pero para la imagen, no. Tenemos que recargar. Solucionemos esto:



Dejamos vacío la img.

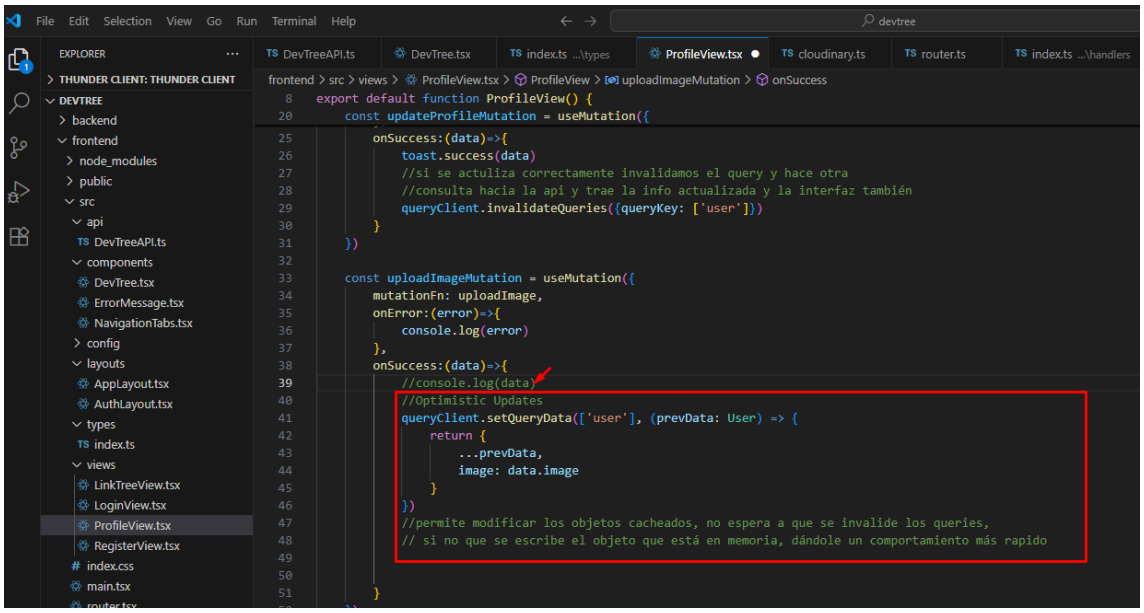




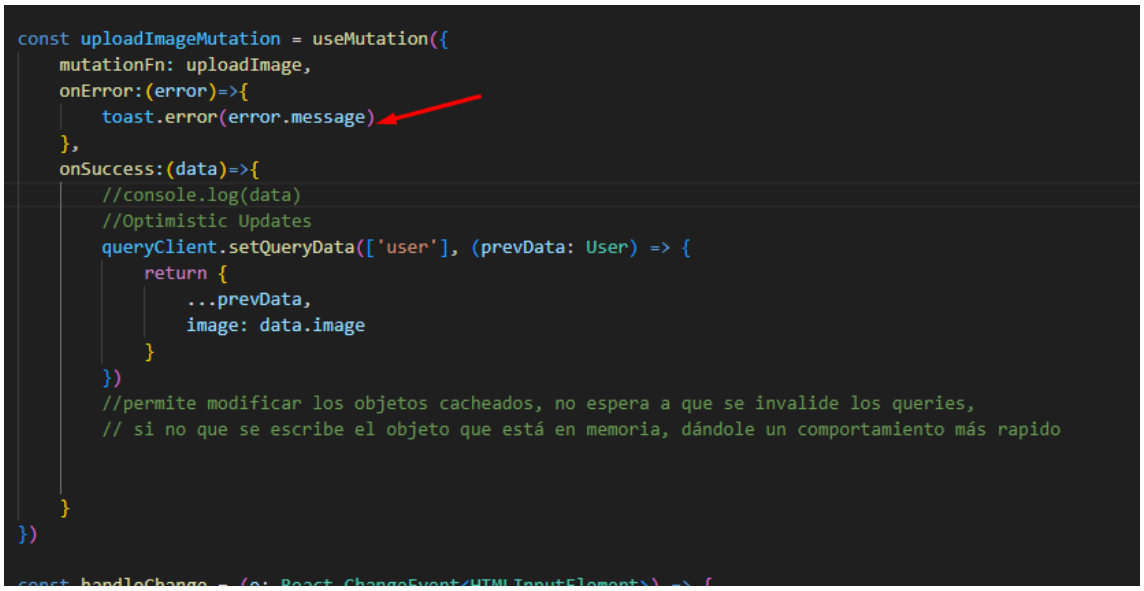
Sube la imagen automático, pero tarda un poco, en lo que se sube la imagen, invalida el query y trae la info actualizada, podemos modificar con algo que se llama Queries Optimistas, es decir, tú sabes lo que va a pasar y puedes adelantarte un poco.

Actualizaciones Optimistas con React Query

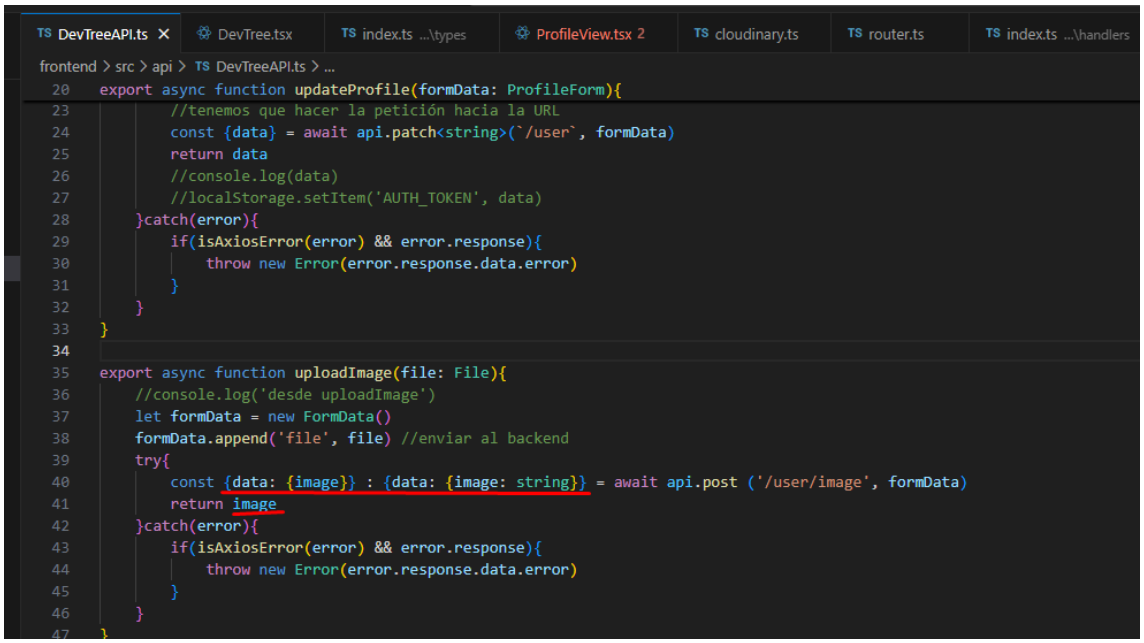
Sabemos que se va a hacer bien la consulta, entonces nos adelantamos escribiendo algo de código que lo haga de forma inmediata. Entonces, en lugar de invalidar los queries

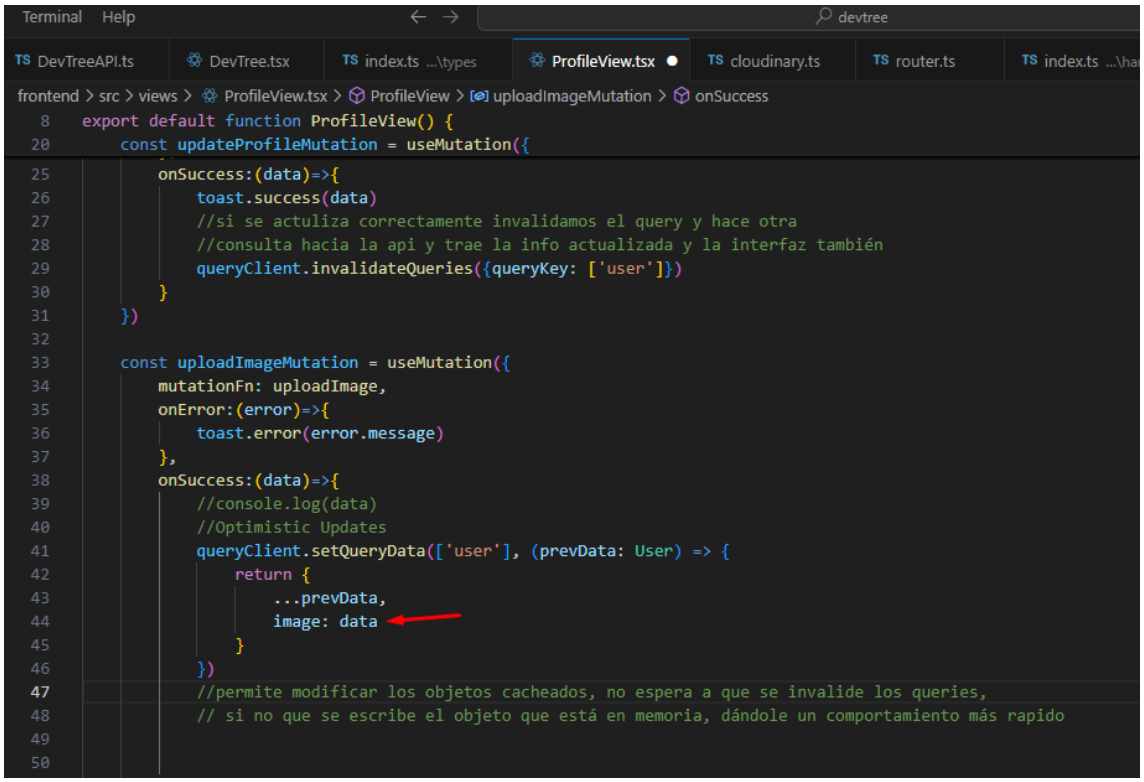


Probamos subiendo una imagen, borramos el url de la imagen desde postman y probamos nuevamente.



El data de aquí viene como “any”, no le podemos poner string, entonces:





Tenemos lista la sección de “Mi Perfil”; ahora pasamos a la sección de Links, que es la parte más compleja.