

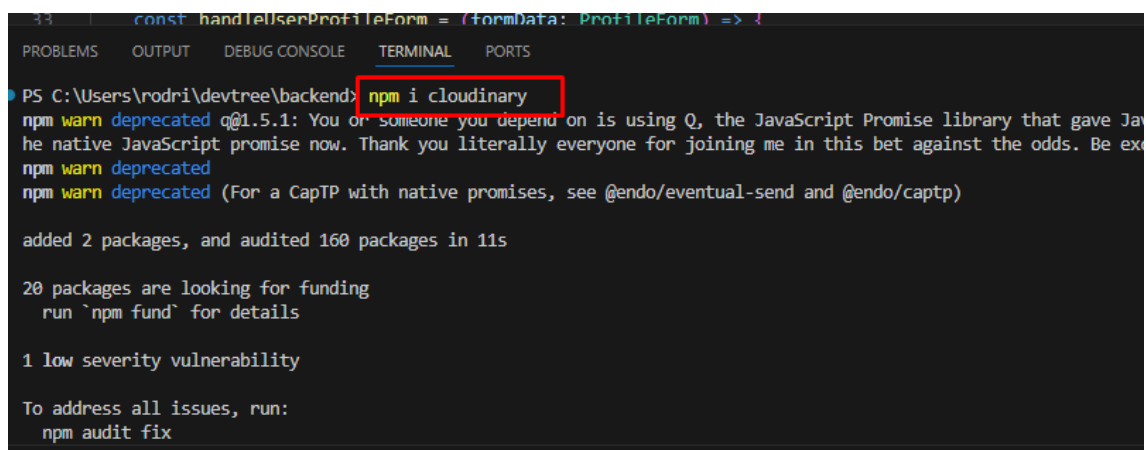
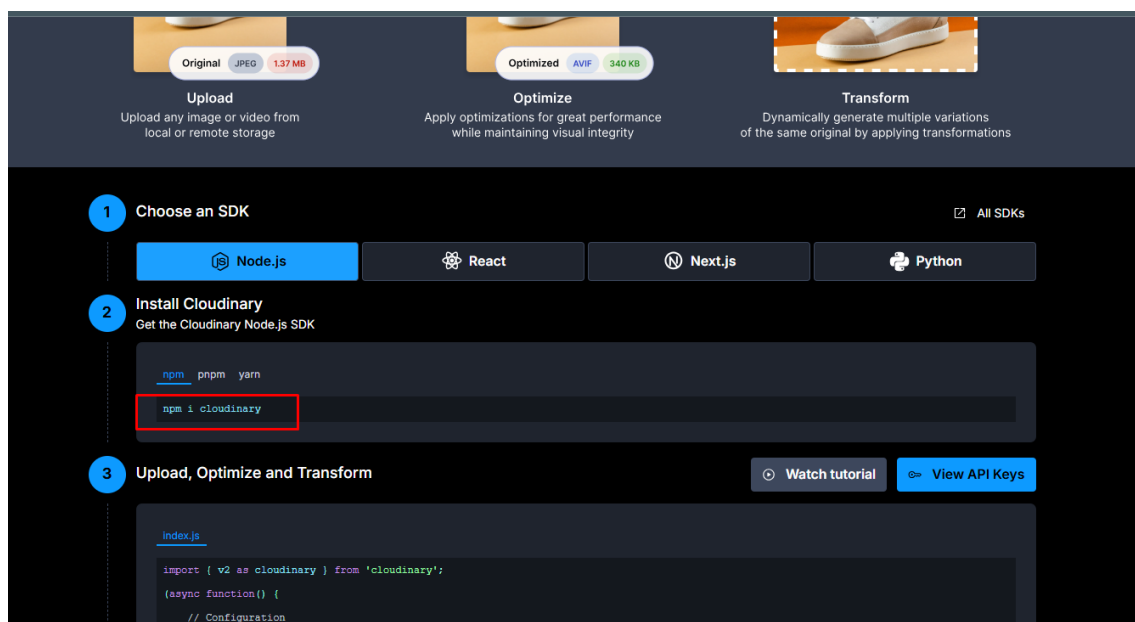
## Subir Imágenes de perfil

### 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> 

```

```

Index.js

import { v2 as cloudinary } from 'cloudinary';

(async function() {
  // Configuration
  cloudinary.config({
    cloud_name: 'daxvaf2cz4',
    api_key: '376398733654435',
    api_secret: '<your_api_secret>' // Click 'View API Keys' above to copy your API secret
  });

  // Upload an image
  const uploadResult = await cloudinary.uploader
    .upload(
      'https://res.cloudinary.com/demo/image/upload/getting-started/shoes.jpg', {
        public_id: 'shoes',
      }
    )
    .catch((error) => {
      console.log(error);
    });

  console.log(uploadResult);

  // Optimize delivery by resizing and applying auto-format and auto-quality
  const optimizeUrl = cloudinary.url('shoes', {
    fetch_format: 'auto',
    quality: 'auto'
  });

  console.log(optimizeUrl);

  // Transform the image: auto-crop to square aspect_ratio
  const autoCropUrl = cloudinary.url('shoes', {
    crop: 'auto',
    gravity: 'auto',
    width: 500,
    height: 500,
  });

  console.log(autoCropUrl);
})();

```

## Agregando las credenciales de Cloudinary en el backend

Crear cloudinary.ts y Copiar:

```

1 import { v2 as cloudinary } from 'cloudinary';
2
3 (async function() {
4
5   // Configuration
6   cloudinary.config({
7     cloud_name: 'dxvaf2cz4',
8     api_key: '376398733654435',
9     api_secret: '<your_api_secret>' // Click 'View API Keys' above to copy your API secret
10  });
11
12  // Upload an image
13  const uploadResult = await cloudinary.uploader
14    .upload(
15    'https://res.cloudinary.com/demo/image/upload/getting-started/shoes.jpg', {
16      public_id: 'shoes',
17    }
18  )
19    .catch((error) => {
20      console.log(error);
21    });
22
23  console.log(uploadResult);
24
25  // Optimize delivery by resizing and applying auto-format and auto-quality
26  const optimizeUrl = cloudinary.url('shoes', {
27    fetch_format: 'auto',
28    quality: 'auto'
29  });
30
31  console.log(optimizeUrl);
32
33  // Transform the image: auto-crop to square aspect_ratio

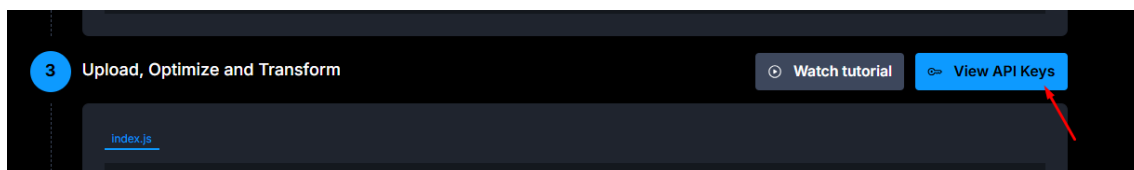
```

```

26 const optimizeUrl = cloudinary.url('shoes', {
27   fetch_format: 'auto',
28   quality: 'auto'
29 });
30
31 console.log(optimizeUrl);
32
33 // Transform the image: auto-crop to square aspect_ratio
34 const autoCropUrl = cloudinary.url('shoes', {
35   crop: 'auto',
36   gravity: 'auto',
37   width: 500,
38   height: 500,
39 });
40
41 console.log(autoCropUrl);
42 });
43
44 export default cloudinary

```

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



```

1 MONGO_URI = mongodb+srv://root:gxNFTdciny$_A19@cluster0.dk0n3nr.mongodb.net/linktree_node_typescript
2 FRONTEND_URL = http://localhost:5173
3 JWT_SECRET = palabrassuperserpetas
4 CLOUDINARY_NAME = dxvaf2cz4
5 CLOUDINARY_API_KEY = 376398733654435
6 CLOUDINARY_API_SECRET = 8AIP1z39B2Xy9kKhvUYNz2ucVFI

```

```

1 import { v2 as cloudinary } from 'cloudinary';
2
3 (async function() {
4
5   // Configuration
6   cloudinary.config({
7     cloud_name: process.env.CLOUDINARY_NAME,
8     api_key: process.env.CLOUDINARY_API_KEY,
9     api_secret: process.env.CLOUDINARY_API_SECRET // Click 'View API Keys' above to copy your API secret
10  });
11

```

## Leyendo imágenes con Formidable

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

```

79 export const updateProfile = async(req: Request, res: Response) => {
80   req.user.handle = handle
81   await req.user.save()
82   res.send('Perfil actualizado correctamente')
83 }
84 catch(e){
85   const error = new Error("Hubo un error")
86   return res.status(500).json({error: error.message})
87 }
88
89 export const uploadImage = async(req: Request, res: Response) => {
90   try{
91   }catch(e){
92     const error = new Error("Hubo un error")
93     return res.status(500).json({error: error.message})
94   }
95 }

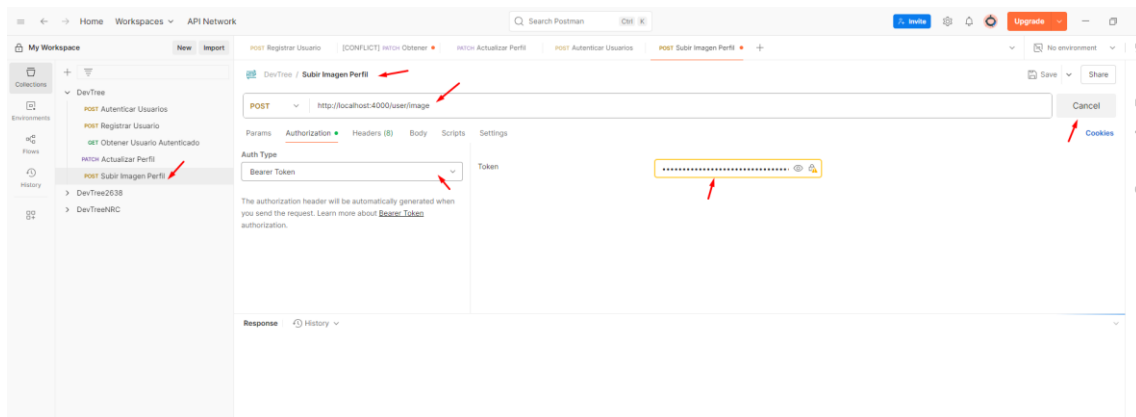
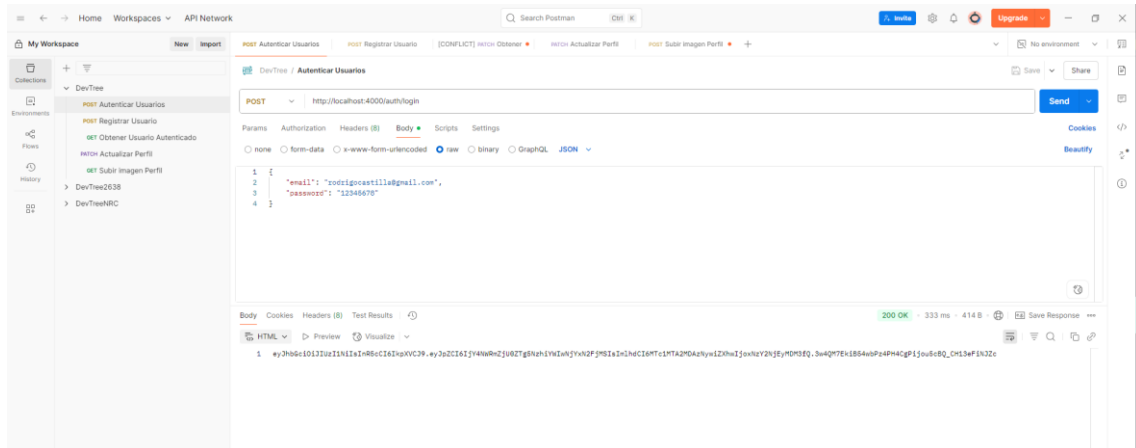
```

```

27 //
28 //withMessage( 'El password es obligatorio '),
29
30 login
31 )
32
33 router.get('/user', authenticate, getUser)
34
35 //actualizar un registro en la BD: put y patch
36 //put: crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos
37 //patch: aplica modificaciones parciales a un recurso (sería mejor porque solo vamos a cambiar el
38
39 //para modificar un usuario necesitamos que esté autenticado por eso el authtenticate.
40 router.patch('/user',
41   //validación:
42   body('handle').notEmpty().withMessage('El handle no puede ir vacío'),
43   body('description').notEmpty().withMessage('La descripción no puede ir vacía'),
44   //el middleware
45   handleInputErrors,
46   authenticate,
47   updateProfile)
48
49 router.post('/user/image', authenticate, uploadImage)
50
51 export default router
52
53

```

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.

```

File Edit Selection View Go Run Terminal Help
EXPLORER
THUNDER CLIENT: THUNDER CLIENT
DEV TREE
  backend
  dist
  node_modules
  src
    config
    TS cloudinary.ts
    TS cors.ts
    TS db.ts
    handlers
      TS index.ts
    middleware
    models
  TS DevTreeAPI.ts
  ProfileView.tsx
  TS cloudinary.ts
  TS routers.ts
  TS index.ts
  .env

backend > src > handlers > TS index.ts > ...
1 import {Request, Response} from 'express'
2 import {validationResult} from 'express-validator'
3 import slug from 'slug'
4 import formidable from './formidable'
5 import jwt from 'jsonwebtoken'
6 import User from '../models/User'
7 import { checkPassword, hashPassword } from '../utils/auth'
8 import { generateJWT } from '../utils/jwt'
9 import cloudinary from '../config/cloudinary'
10
11 export const createAccount = async(req: Request, res: Response)=>{
12
13   //console.log(req.body)
14
15   const {email, password} = req.body

```

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

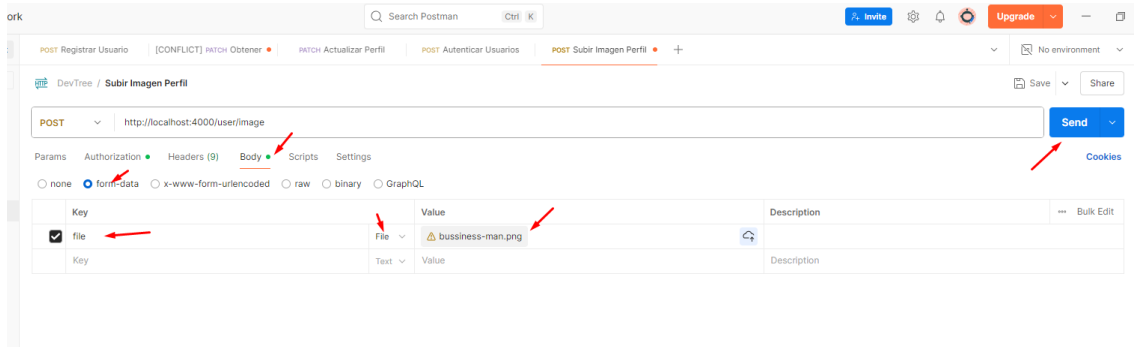
Run `npm audit` for details.
PS C:\Users\rodri\devtree\backend>

```

```

81 export const updateProfile = async(req: Request, res: Response) => {
82
83 }
84 //Actualizar el usuario
85 req.user.description = description
86 req.user.handle = handle
87
88 await req.user.save()
89 res.send('Perfil actualizado correctamente')
90
91 }catch(e){
92   const error = new Error("Hubo un error")
93   return res.status(500).json({error: error.message})
94 }
95
96 export const uploadImage = async(req: Request, res: Response) => {
97
98   const form = formidable({multiples: false}) //porque solo subiremos una imagen
99   form.parse(req, (error, fields, files)=>{
100     console.log(files)
101   }) //leyendo datos
102
103   try{
104     console.log('Desde uploadImage')
105   }catch(e){
106     const error = new Error("Hubo un error")
107     return res.status(500).json({error: error.message})
108   }
109 }
110
111 }
112
113
114
115
116
117
118
119
120

```



```

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\g33kk0ydz7ynjh1p0l0c0a67',
      newFilename: 'g33kk0ydz7ynjh1p0l0c0a67',
      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")
  }
}
```

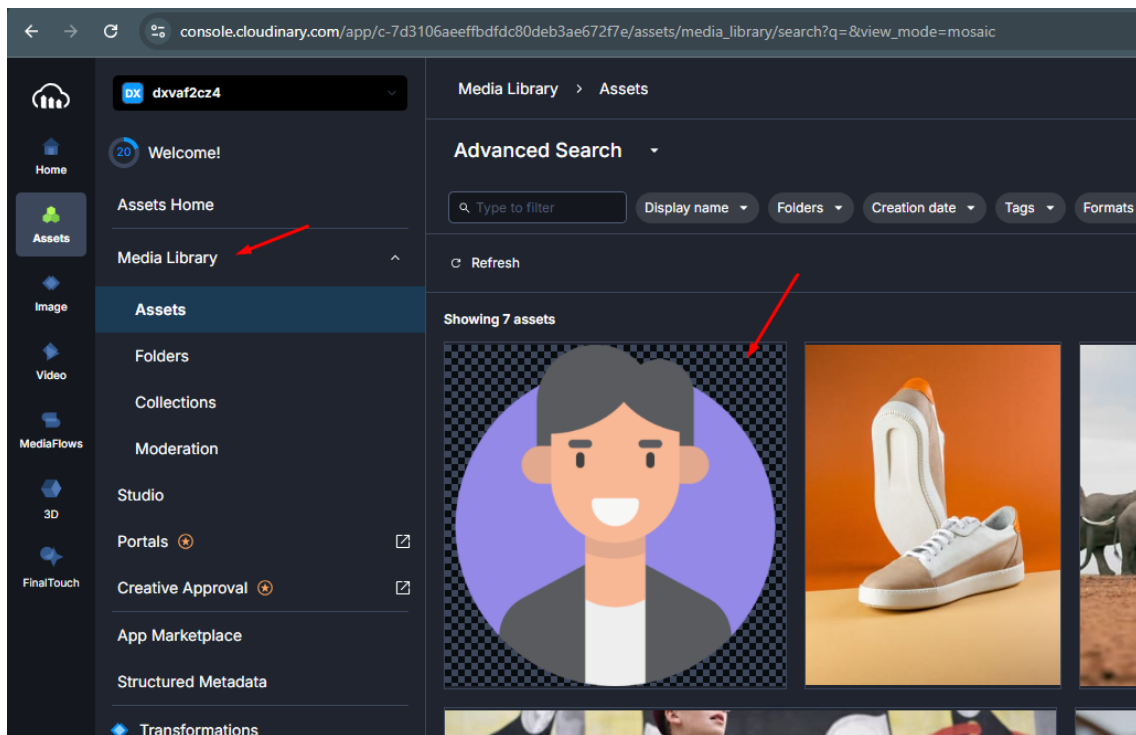
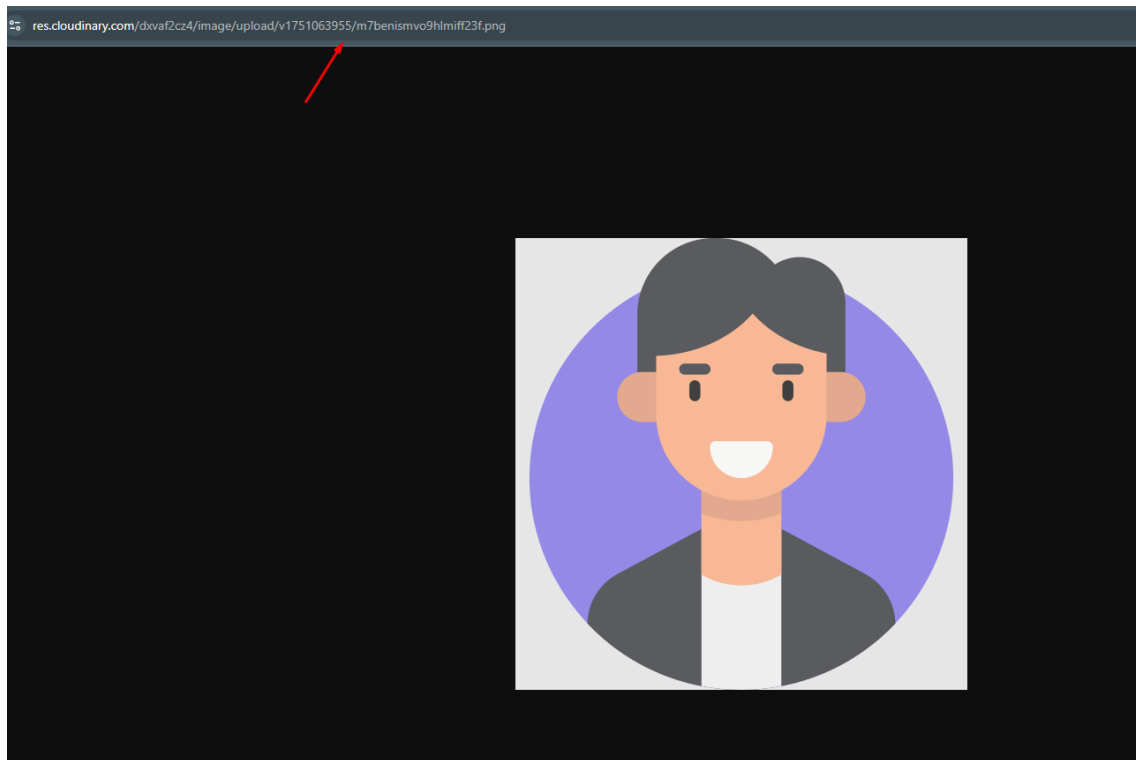
Reiniciamos servidor, enviamos nuevamente desde postman.

```
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: '7b492df6877444dddcaddc8a0f72e23',
  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/t_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/ft3jadfvmfwlmvtekmi.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: Uniqueld

```
● 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

```

TS DevTreeAPI.ts  ProfileView.tsx  TS cloudinary.ts  TS router.ts  TS index.ts  TS User.ts  .env
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: ''
  }
})

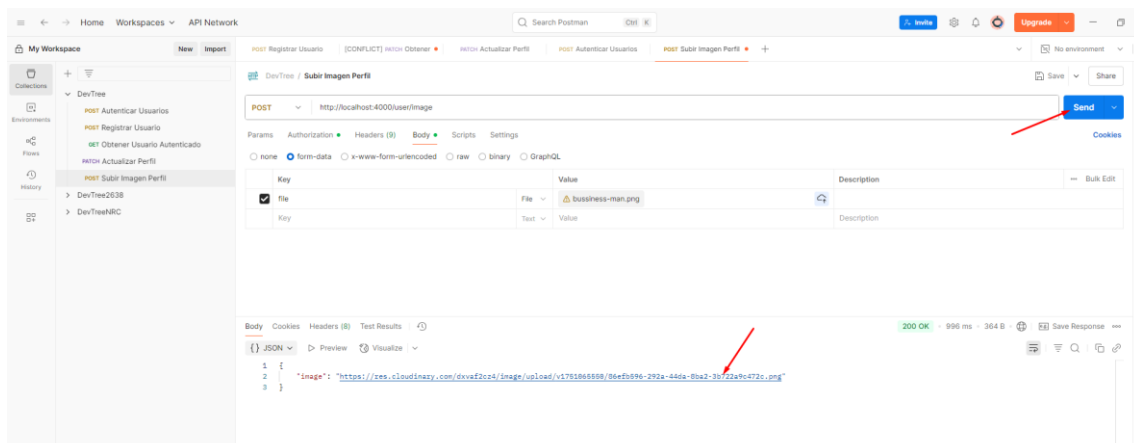
```

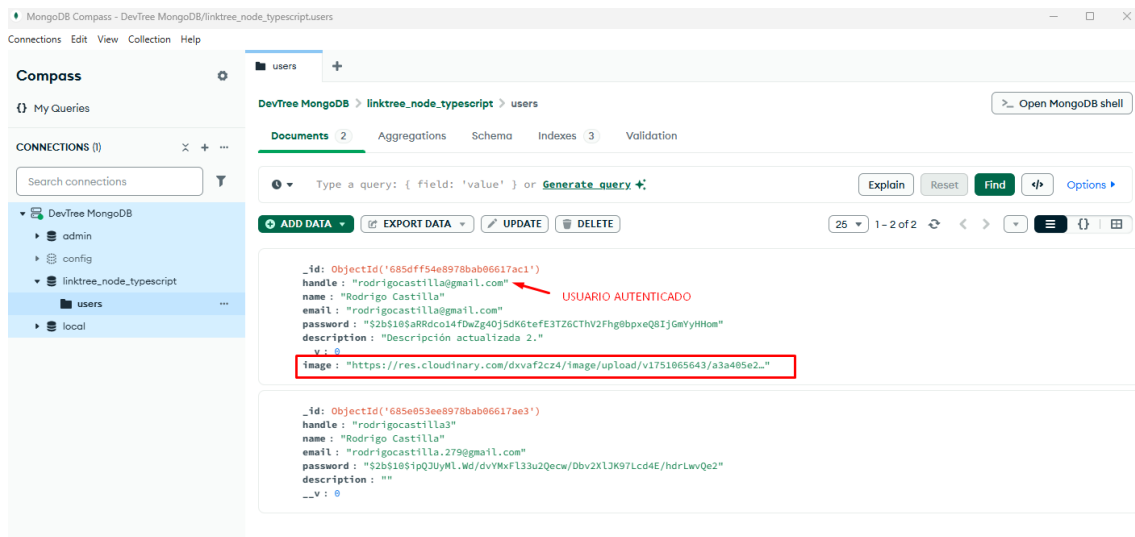
## Index.ts

```

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.iamege = result.secure_url
        await req.user.save()
        res.json({image: result.secure_url})
      }
    })
  })
}

```

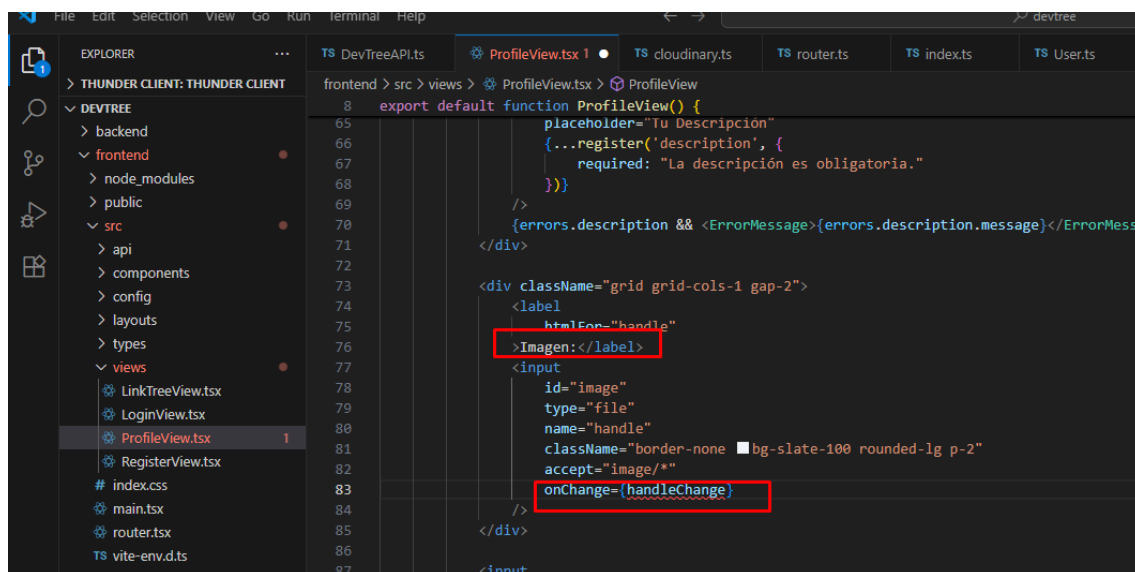




Entonces ya la podemos renderizar.

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

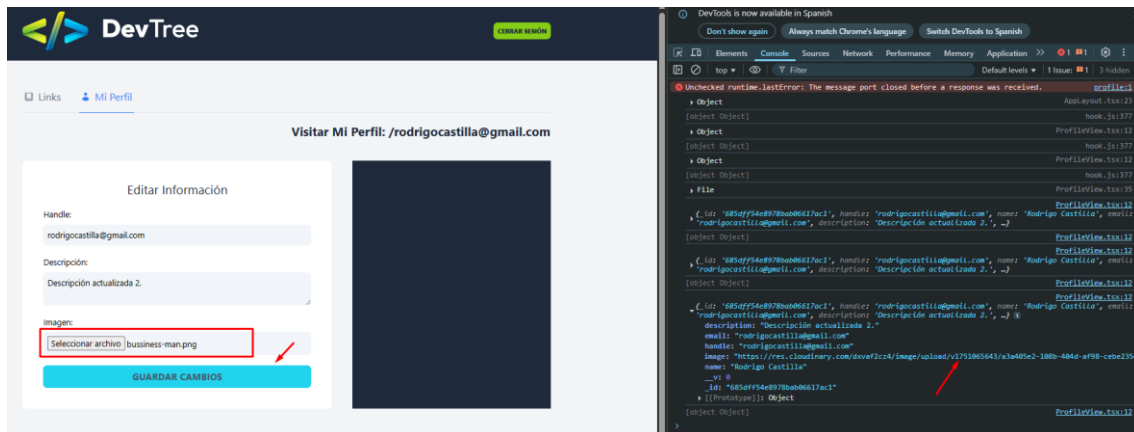
### Leyendo el archivo que se sube en React



```

TS DevTreeAPI.ts  ProfileView.tsx  TS cloudinary.ts  TS router.ts  TS index.ts  TS User.ts  .env
frontend > src > views > ProfileView.tsx > ProfileView > handleChange
8  export default function ProfileView() {
16    {handle: data.handle,
17     description: data.description
18    }}
19
20    const updateProfileMutation = useMutation({
21      mutationFn: updateProfile,
22      onError: (error) => {
23        toast.error(error.message)
24      },
25      onSuccess: (data) => {
26        toast.success(data)
27        //si se actualiza correctamente invalidamos el query y hace otra
28        //consulta hacia la api y trae la info actualizada y la interfaz también
29        queryClient.invalidateQueries({queryKey: ['user']})
30      }
31    })
32
33    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
34      if(e.target.files){ //puede ser que el usuario haga el cambio pero no suba el archivo
35        console.log(e.target.files[0])
36      }
37    }
38
39    const handleUserProfileForm = (formData: ProfileForm) => {
40      //console.log(formData)
41      updateProfileMutation.mutate(formData)

```



Enviando la imagen con Axios y ReactQuery

```

20 export async function updateProfile(formData: ProfileForm){
21
22   try{
23     //tenemos que hacer la petición hacia la URL
24     const {data} = await api.patch<string>(`/user`, formData)
25     return data
26     //console.log(data)
27     //localStorage.setItem('AUTH_TOKEN', data)
28   }catch(error){
29     if(isAxiosError(error) && error.response){
30       throw new Error(error.response.data.error)
31     }
32   }
33 }
34
35 export async function uploadImage(){
36   console.log('desde uploadImage')
37   try{
38
39   }catch(error){
40     if(isAxiosError(error) && error.response){
41       throw new Error(error.response.data.error)
42     }
43   }
44 }

```

Ahora hay que crear la mutación.

```

8 export default function ProfileView() {
20   const updateProfileMutation = useMutation({
22     onError: (error) => {
23       toast.error(error.message)
24     },
25     onSuccess: (data) => {
26       toast.success(data)
27       //si se actualiza correctamente invalidamos el query y hace otra
28       //consulta hacia la api y trae la info actualizada y la interfaz también
29       queryClient.invalidateQueries({queryKey: ['user']})
30     }
31   })
32
33   const uploadImageMutation = useMutation({
34     mutationFn: uploadImage,
35     onError: (error) => {
36       console.log(error)
37     },
38     onSuccess: (data) => {
39       console.log(data)
40     }
41   })
42
43   const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
44     if(e.target.files){ //puede ser que el usuario haga el cambio pero no suba el archivo
45       uploadImageMutation.mutate(e.target.files[0])
46     }
47   }
48 }

```

COPIAR FUNCIÓN DE ARRIBA

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

```

    TS DevTreeAPI.ts
    frontend > src > api > TS DevTreeAPI.ts > uploadImage

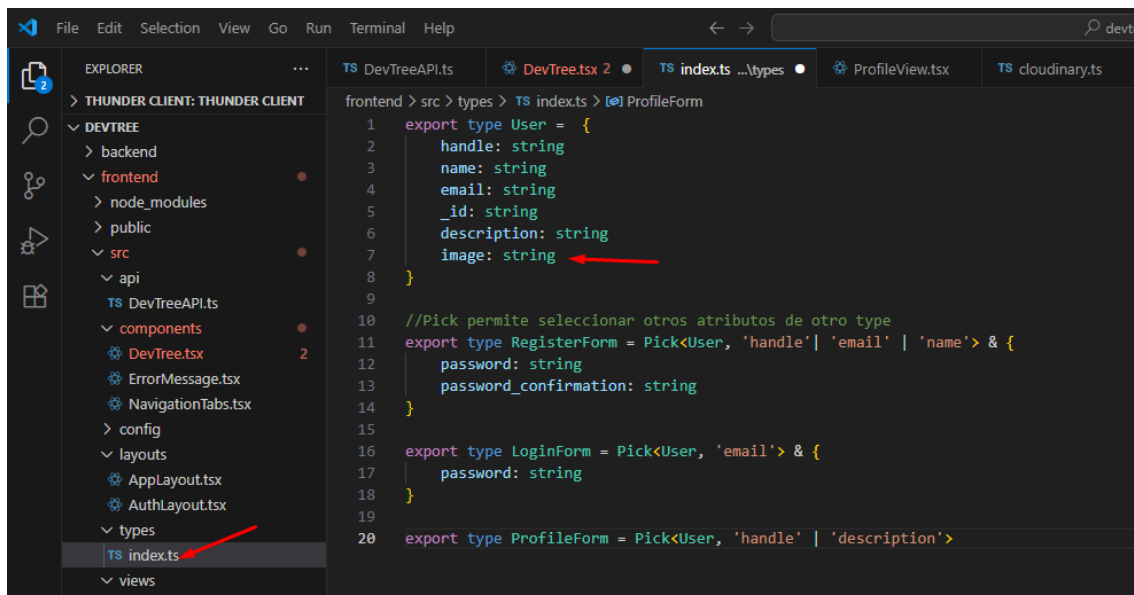
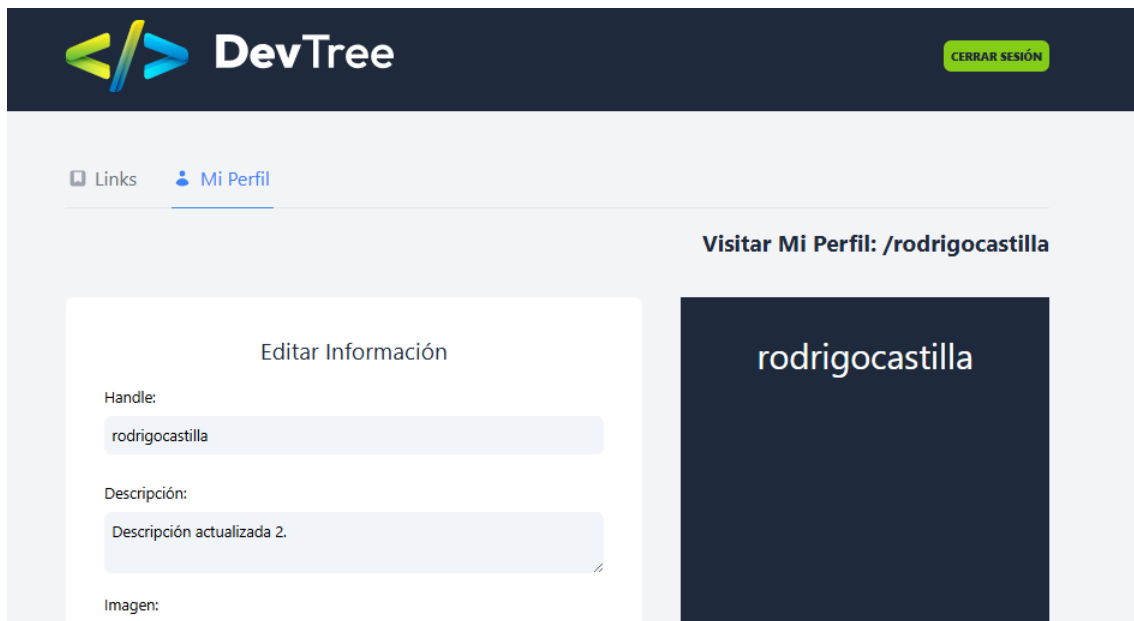
    20 export async function updateProfile(formData: ProfileForm){
    21
    22     try{
    23         //tenemos que hacer la petición hacia la URL
    24         const {data} = await api.patch<string>('/user', formData)
    25         return data
    26         //console.log(data)
    27         //localStorage.setItem('AUTH_TOKEN', data)
    28     }catch(error){
    29         if(isAxiosError(error) && error.response){
    30             throw new Error(error.response.data.error)
    31         }
    32     }
    33 }
    34
    35 export async function uploadImage(file: File){
    36     //console.log('desde uploadImage')
    37     let formData = new FormData()
    38     formData.append('file', file) //enviar al backend
    39     try{
    40         const {data} = await api.post('/user/image', formData)
    41         return data
    42     }catch(error){
    43         if(isAxiosError(error) && error.response){
    44             throw new Error(error.response.data.error)
    45         }
    46     }
    47 }
  
```

## Mostrando la información de Perfil e Imagen

```

    TS DevTreeAPI.ts
    frontend > src > components > DevTree.tsx > DevTree

    10 export default function DevTree({data}: DevTreeProps){
    11
    12     </div>
    13     </header>
    14     <div className="bg-gray-100 min-h-screen py-10">
    15         <main className="mx-auto max-w-5xl p-10 md:p-0">
    16
    17             <NavigationTabs />
    18
    19             <div className="flex justify-end">
    20                 <Link
    21                     className="font-bold text-right text-slate-800 text-2xl"
    22                     to={}
    23                     target="_blank"
    24                     rel="noreferrer noopener"
    25                     >Visitar Mi Perfil: {data.handle}</Link>
    26             </div>
    27
    28             <div className="flex flex-col md:flex-row gap-10 mt-10">
    29                 <div className="flex-1">
    30                     <Outlet />
    31                 </div>
    32                 <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
    33                     <p className="text-4xl text-center text-white">{data.handle}</p>
    34                 </div>
    35             </div>
    36         </main>
    37     </div>
    38 }
    39
    40
    41
    42
    43
    44
    45
    46
    47
    48
    49
    50
    51
  
```





```

TS DevTreeAPI.ts  DevTree.tsx X TS index.ts ...types  ProfileView.tsx  TS cloudinary.ts  TS router.ts  TS index.ts ...handlers  TS Users.ts

frontend > src > components > DevTree.tsx > DevTree
10 export default function DevTree({data}: DevTreeProps){
26   </div>
27   </header>
28   <div className="bg-gray-100 min-h-screen py-10">
29     <main className="mx-auto max-w-5xl p-10 md:p-0">
30
31       <NavigationTabs />
32
33       <div className="flex justify-end">
34         <Link
35           className="font-bold text-right text-slate-800 text-2xl"
36           to={' '}
37           target="_blank"
38           rel="noopener noreferrer"
39         >Visitar Mi Perfil: {data.handle}</Link>
40       </div>
41
42       <div className="flex flex-col md:flex-row gap-10 mt-10">
43         <div className="flex-1">
44           <Outlet />
45         </div>
46         <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
47           <p className="text-4xl text-center text-white">{data.handle}</p>
48           <img src={data.image} alt="Imagen Perfil" className="mx-auto max-w-[250px]" />
49         </div>
50       </div>
51     </main>
52   </div>
53   <Toaster position="top-right" />
54 </>
55

```

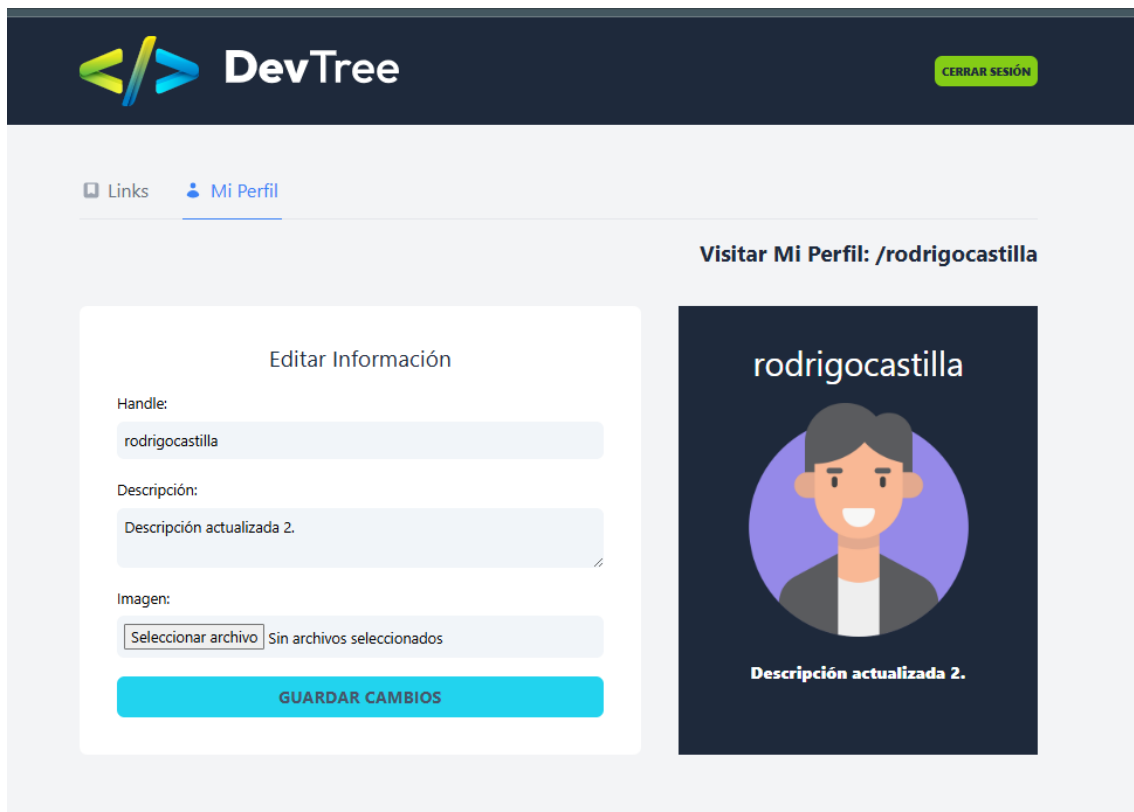
Para usuarios que no tengan foto, les saldrá un enlace roto. Lo arreglamos:

```

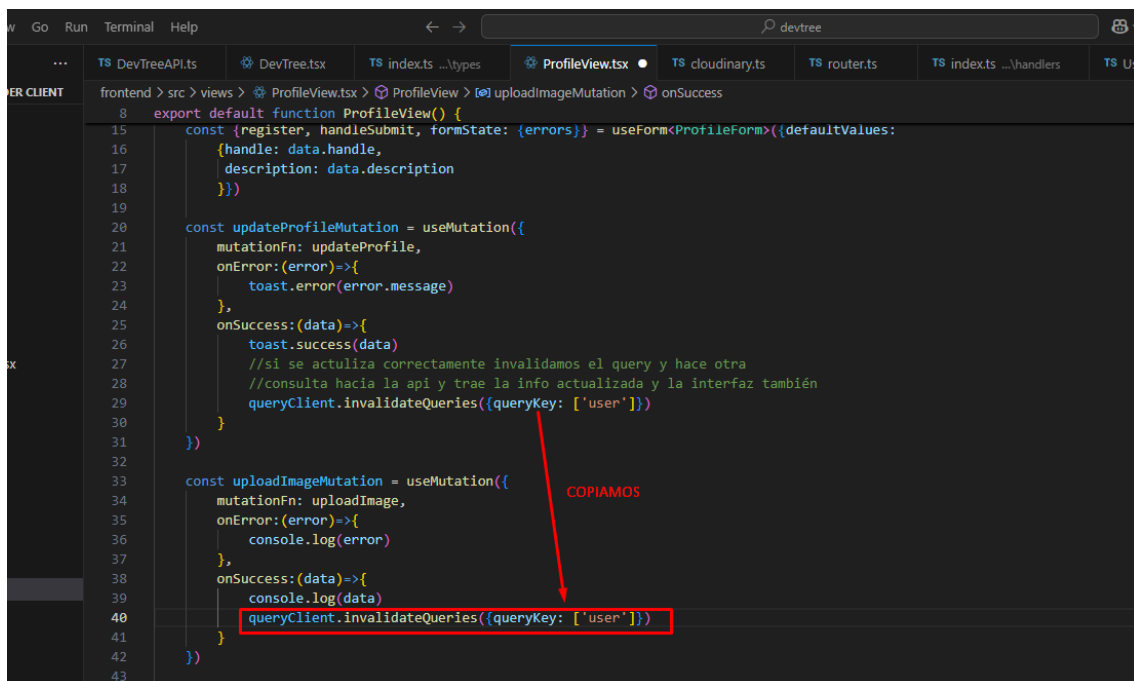
TS DevTreeAPI.ts  DevTree.tsx X TS index.ts ...types  ProfileView.tsx  TS cloudinary.ts  TS router.ts  TS index.ts ...handlers  TS Users.ts

frontend > src > components > DevTree.tsx > DevTree
10 export default function DevTree({data}: DevTreeProps){
26   </div>
27   </header>
28   <div className="bg-gray-100 min-h-screen py-10">
29     <main className="mx-auto max-w-5xl p-10 md:p-0">
30
31       <NavigationTabs />
32
33       <div className="flex justify-end">
34         <Link
35           className="font-bold text-right text-slate-800 text-2xl"
36           to={' '}
37           target="_blank"
38           rel="noopener noreferrer"
39         >Visitar Mi Perfil: {data.handle}</Link>
40       </div>
41
42       <div className="flex flex-col md:flex-row gap-10 mt-10">
43         <div className="flex-1">
44           <Outlet />
45         </div>
46         <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
47           <p className="text-4xl text-center text-white">{data.handle}</p>
48           {data.image && //si tenemos una imagen, la renderizamos, pero si no:
49           <img src={data.image} alt="Imagen Perfil" className="mx-auto max-w-[250px]" />
50           }
51           <p className="text-center text-lg font-black text-white">{data.description}</p>
52         </div>
53       </div>
54     </main>
55   </div>
56   <Toaster position="top-right" />
57 </>

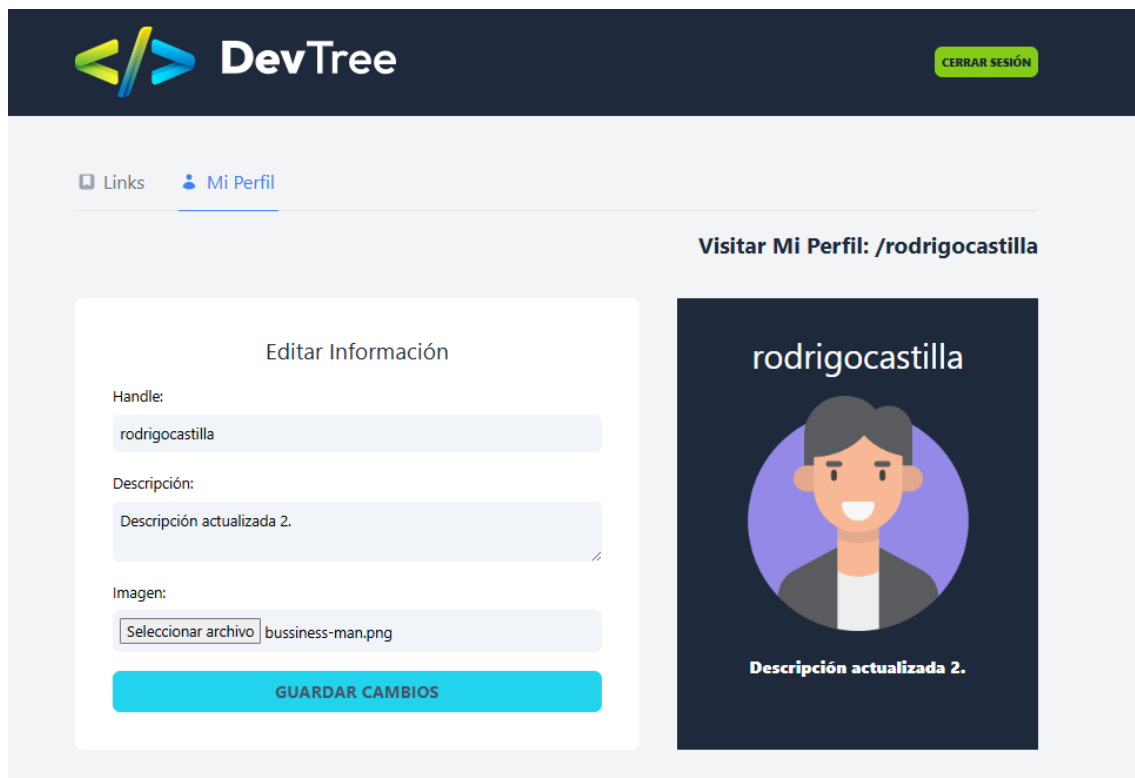
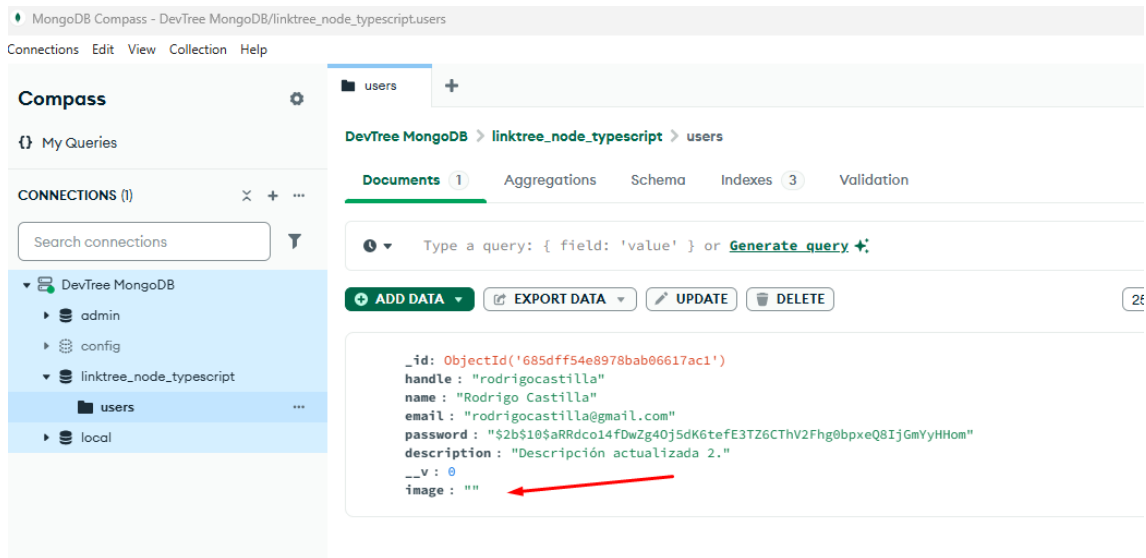
```



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 automáticamente, 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

```

8  export default function ProfileView() {
20  const updateProfileMutation = useMutation({
25      onSuccess:(data)=>{
26          toast.success(data)
27          //si se actualiza correctamente invalidamos el query y hace otra
28          //consulta hacia la api y trae la info actualizada y la interfaz también
29          queryClient.invalidateQueries({queryKey: ['user']})
30      }
31  })
32
33  const uploadImageMutation = useMutation({
34      mutationFn: uploadImage,
35      onError:(error)=>{
36          console.log(error)
37      },
38      onSuccess:(data)=>{
39          //console.log(data)
40          //Optimistic Updates
41          queryClient.setQueryData(['user'], (prevData: User) => {
42              return {
43                  ...prevData,
44                  image: data.image
45              }
46          })
47          //permite modificar los objetos cacheados, no espera a que se invalide los queries,
48          // si no que se escribe el objeto que está en memoria, dándole un comportamiento más rapido
49      }
50  })
51

```

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

```

const uploadImageMutation = useMutation({
  mutationFn: uploadImage,
  onError:(error)=>{
    toast.error(error.message)
  },
  onSuccess:(data)=>{
    //console.log(data)
    //Optimistic Updates
    queryClient.setQueryData(['user'], (prevData: User) => {
      return {
        ...prevData,
        image: data.image
      }
    })
    //permite modificar los objetos cacheados, no espera a que se invalide los queries,
    // si no que se escribe el objeto que está en memoria, dándole un comportamiento más rapido
  }
})

```

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

```

TS DevTreeAPI.ts X DevTree.tsx TS index.ts ...types ProfileView.tsx 2 TS cloudinary.ts TS router.ts TS index.ts ...handlers
frontend > src > api > TS DevTreeAPI.ts > ...
20 export async function updateProfile(formData: ProfileForm){
23   //tenemos que hacer la petición hacia la URL
24   const {data} = await api.patch<string>(`/user`, formData)
25   return data
26   //console.log(data)
27   //localStorage.setItem('AUTH_TOKEN', data)
28 }catch(error){
29   if(isAxiosError(error) && error.response){
30     throw new Error(error.response.data.error)
31   }
32 }
33 }
34
35 export async function uploadImage(file: File){
36   //console.log('desde uploadImage')
37   let formData = new FormData()
38   formData.append('file', file) //enviar al backend
39   try{
40     const {data: {image}} : {data: {image: string}} = await api.post(`/user/image`, formData)
41     return image
42   }catch(error){
43     if(isAxiosError(error) && error.response){
44       throw new Error(error.response.data.error)
45     }
46   }
47 }

```

```

Terminal Help < -> devtree
TS DevTreeAPI.ts DevTree.tsx TS index.ts ...types ProfileView.tsx TS cloudinary.ts TS router.ts TS index.ts ...handlers
frontend > src > views > ProfileView.tsx > ProfileView > uploadImageMutation > onSuccess
8 export default function ProfileView() {
20   const updateProfileMutation = useMutation({
25     onSuccess:(data)=>{
26       toast.success(data)
27       //si se actualiza correctamente invalidamos el query y hace otra
28       //consulta hacia la api y trae la info actualizada y la interfaz también
29       queryClient.invalidateQueries({queryKey: ['user']})
30     }
31   })
32
33   const uploadImageMutation = useMutation({
34     mutationFn: uploadImage,
35     onError:(error)=>{
36       toast.error(error.message)
37     },
38     onSuccess:(data)=>{
39       //console.log(data)
40       //Optimistic Updates
41       queryClient.setQueryData(['user'], (prevData: User) => {
42         return {
43           ...prevData,
44           image: data
45         }
46       })
47       //permite modificar los objetos cacheados, no espera a que se invalide los queries,
48       // si no que se escribe el objeto que está en memoria, dándole un comportamiento más rapido
49     }
50   })

```

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