

Panel para Administrar Perfil

Creando el formulario y diferencias entre autenticación y autorización.

Los formularios deben precargar la información, el backend aún no sabe quién soy yo, quién está visitando el sitio web. ¿Cómo lo va a identificar?

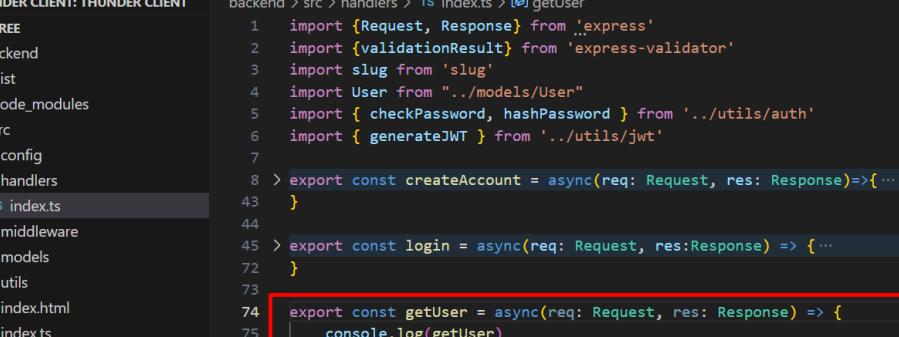
Creando un endpoint en el backend para obtener el usuario autenticado

Cuando iniciamos sesión obtenemos el JWT y lo almacenamos en Local Storage, pero tenemos que enviar el JWT y el servidor tiene que validararlo.

Creamos un nuevo request en Postman.

Vamos al backend y abrimos el router y los handlers.

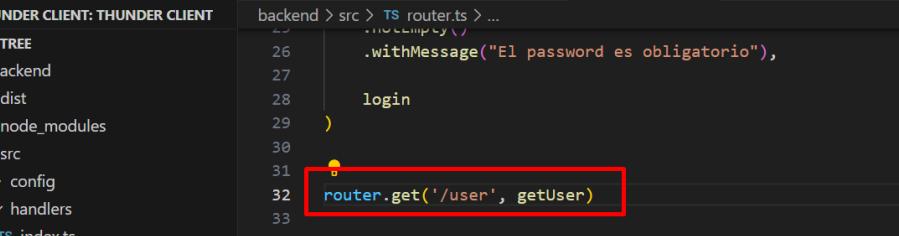
Creamos un handler. `Console.log('Desde getUser')`



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "DEVTREE". The "backend" folder is expanded, revealing "dist", "node_modules", "src", "config", "handlers", and several files like "index.html", "index.ts", "router.ts", and "server.ts".
- Code Editor (Right):** The current file is "ProfileView.tsx" (highlighted in blue), but the code shown is from "index.ts". The code defines an asynchronous function "getUser" that takes a Request and Response object, logs the result to the console, and returns it.
- Status Bar:** Shows the path "backend > src > handlers > index.ts" and the line number "77".
- Search Bar:** Contains the text "devtree".

```
1 import {Request, Response} from 'express'
2 import {validationResult} from 'express-validator'
3 import slug from 'slug'
4 import User from '../models/User'
5 import { checkPassword, hashPassword } from '../utils/auth'
6 import { generateJWT } from '../utils/jwt'
7
8 > export const createAccount = async(req: Request, res: Response) => {
9
10 }
11
12 > export const login = async(req: Request, res: Response) => {
13
14 }
15
16 > export const getUser = async(req: Request, res: Response) => {
17   |   console.log(getUser)
18 }
```



```
backend > src > TS router.ts > ...
26     .withMessage("El password es obligatorio"),
27
28     login
29 )
30
31 router.get('/user', getUser)
32
33
34 export default router
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with tabs for 'Collections', 'Environments', 'Flows', and 'History'. The main workspace is titled 'My Workspace' and contains a collection named 'DevTree'. Under 'DevTree', there are three requests: 'POST Autenticar Usuarios', 'POST Registrar Usuario', and 'GET Obtener Usuario Autenticado'. The 'GET' request is currently selected. The request details panel shows a 'GET' method pointing to 'http://localhost:4000/user'. Below the method, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Scripts', and 'Settings'. The 'Params' tab is active, showing a table with one row and columns for 'Key' and 'Value'. The 'Headers' tab shows eight entries. At the bottom, there are tabs for 'Body', 'Cookies', 'Headers (8)', 'Test Results', and a refresh icon. Below these tabs, there are buttons for 'HTML', 'Preview', and 'Visualize'. On the right side of the interface, there are buttons for 'Save', 'Share', and 'Cancel'. A top navigation bar includes 'Home', 'Workspaces', 'API Network', a search bar for 'Postman', and buttons for 'Invite', 'Upgrade', and environment management.

El lugar donde se coloca el JWT es en los headers. En postman tenemos esa pestaña de Headers que se envían antes del body pero se envían todo junto. También tenemos Authorization, hay diferentes tipos. Para enviar el JWT utilizaremos Bearer

My Workspace

POST Autenticar Usuarios • POST Registrar Usuario • GET Obtener Usuario Autent. • +

HTTP DevTree / Obtener Usuario Autenticado

GET http://localhost:4000/user

Params Authorization • Headers (6) Body Scripts Settings

Auth Type Bearer Token

Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Send

File Edit Selection View Go Run ...

EXPLORER

THUNDER CLIENT: THUNDER CLIENT

DEVTREE

backend

src

handlers

index.ts

ProfileView.tsx index.ts cors.ts

```

45 export const login = async(req: Request, res: Response) => {
46
47     //Hasta aqui tenemos un usuario que ingresó correctamente, entonces:
48
49     const token = generateJWT({id: user._id})
50     res.send(token)
51
52 }
53
54 export const getUser = async(req: Request, res: Response) => {
55     console.log(req.headers.authorization)
56 }
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
    
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

My Workspace

POST Autenticar Usuarios • POST Registrar Usuario • GET Obtener Usuario Autent. • +

HTTP DevTree / Obtener Usuario Autenticado

GET http://localhost:4000/user

Params Authorization • Headers (7) Body Scripts Settings

Auth Type Bearer Token

Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Cancel Cookies

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

> 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
[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto 4000
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4NWEwNDMyNTk1MTA0M2Y1ZTzi
c40CwizXhwIjoxNzY2MjgzNzg4fQ.Re6XpBr94nXwL8f3sBq25QKjHsNRzXBpgtKGUs6Lyel
[]
```

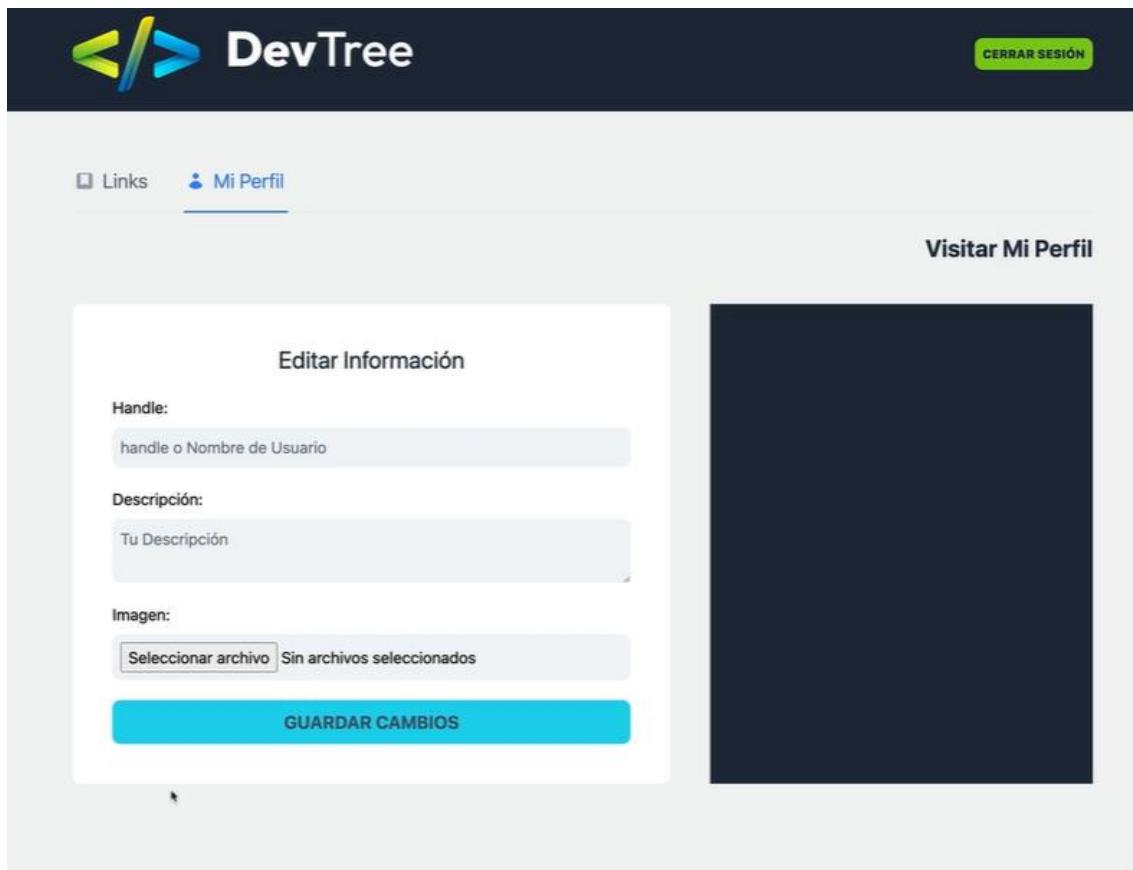
Si en postman enviamos un token vacío. Aparecerá undefined

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

> 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
[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto 4000
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4NWEwNDMyNTk1MTA0M2Y1ZT
c40CwizXhwIjoxNzY2MjgzNzg4fQ.Re6XpBr94nXwL8f3sBq25QKjHsNRzXBpgtKGUs6Lyel
undefined
[]
```

Esa será nuestra primera validación, porque habrán personas que querrán ingresar a esta área donde tienes que tener una cuenta.



```
backend > src > handlers > TS index.ts > [e] getUser
45  export const login = async(req: Request, res: Response) => {
46
47    const token = generateJWT({id: user._id})
48    res.send(token)
49
50  }
51
52
53
54  export const getUser = async(req: Request, res: Response) => {
55    const bearer = req.headers.authorization
56    if(!bearer){
57      const error = new Error('No autorizado')
58      return res.status(401).json({error: error.message})
59    }
60  }
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
```

The screenshot displays the Thunder Client interface with the code editor open. The file 'index.ts' is shown, containing TypeScript code for a 'login' function and a 'getUser' function. The 'getUser' function is highlighted with a red box. The code checks for an authorization header and returns a 401 error if it's missing.

Enviamos una petición con token vacío:

The screenshot shows the Postman interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'DevTree' and environments like 'DevTree2638'. In the main area, a collection named 'DevTree / Obtener Usuario Autenticado' is selected. A GET request is configured with the URL `http://localhost:4000/user`. The 'Authorization' type is set to 'Bearer Token' and the 'Token' field is empty, indicated by a red box. The response status is 401 Unauthorized, and the response body is `{"error": "No autorizado"}`, also highlighted with a red box.

Identificando el usuario autenticado por JWT

Puede ser que haya usuarios que con código manden un Bearer que esté vacío, haremos lo siguiente:

The screenshot shows the VS Code editor with the file `index.ts` open. The code contains logic for handling a Bearer token from the request headers. A red X is drawn over the line `const [, token] = bearer.split(' ')`, indicating where a user might be sending an empty token. The terminal at the bottom shows the node.js server running on port 4000.

```

45 export const login = async(req: Request, res:Response) => {
46   ...
47 }
48
49 export const getUser = async(req: Request, res: Response) => {
50   const bearer = req.headers.authorization
51   if(!bearer){
52     const error = new Error('No autorizado')
53     return res.status(401).json({error: error.message})
54   }
55   const [ , token] = bearer.split(' ')
56 }
57
58
59 
```

The screenshot shows the VS Code editor with the file `index.ts` open. The code imports various modules and defines a `createAccount` function. A red box highlights the import statement `import jwt from 'jsonwebtoken'`. The terminal at the bottom shows the node.js server running on port 4000.

```

1 import {Request, Response} from 'express'
2 import {validationResult} from 'express-validator'
3 import slug from 'slug'
4 import jwt from 'jsonwebtoken'
5 import User from '../models/User'
6 import { checkPassword, hashPassword } from '../utils/auth'
7 import { generateJWT } from '../utils/jwt'
8
9 > export const createAccount = async(req: Request, res: Response)=>{ ...
10 }
11
12 export const login = async(req: Request, res:Response) => {
13   //Manejar errores
14   let errors = validationResult(req)
15   if (!errors.isEmpty()){
16     return res.status(400).json({errors: errors.array()})
17   }
18   const user = await User.findOne({email: req.body.email})
19   if (!user)
20     return res.status(401).json({error: 'Email no encontrado'})
21   const isMatch = await user.checkPassword(req.body.password)
22   if (!isMatch)
23     return res.status(401).json({error: 'Contraseña incorrecta'})
24   const token = jwt.sign({id: user._id, email: user.email}, process.env.JWT_SECRET)
25   res.cookie('token', token, {maxAge: 60000 * 60 * 24, httpOnly: true}).status(200).json({user, token})
26 }
27 
```

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows the project structure under "DEVTREE". Key files include `ProfileView.tsx`, `index.ts` (selected), `cors.ts`, `db.ts`, `handlers`, `middleware`, `models`, `utils`, `auth.ts`, `jwt.ts`, `index.html`, `index.ts`, `router.ts`, `server.ts`, `.env`, `package-lock.json`, `package.json`, and `tsconfig.json`.
- Code Editor:** Displays the `index.ts` file with the following code:

```
73 }
74
75 export const getUser = async(req: Request, res: Response) => {
76   const bearer = req.headers.authorization
77   if(!bearer){
78     const error = new Error('No autorizado')
79     return res.status(401).json({error: error.message})
80   }
81   const [, token] = bearer.split(' ')
82   if(!token){
83     const error = new Error('No autorizado')
84     return res.status(401).json({error: error.message})
85   }
86   try{
87     const result = jwt.verify(token, process.env.JWT_SECRET)
88     console.log(result)
89   }catch(error){
90     res.status(500).json({error: 'Token no válido'})
91   }
92 }
```

- Terminal:** Shows logs from nodemon indicating the server is running on port 4000.
- Status Bar:** Shows the command `[id: '685a04325951043f5e6b7a7d', iat: 1750731788, exp: 1766283788]` which corresponds to the highlighted JWT token in the code editor.

Enviamos una petición desde postman y tenemos el id, que era lo mínimo que se requiere para identificar el usuario. Ahora, tenemos que buscar a ese usuario con ese id, puede que se quede en el navegador, pero quizás es un colaborador que ya no trabaja, entonces hay que revisar que el usuario sea válido.

```

File Edit Selection View Go Run ... ⏪ ⏩ ⏴ ⏵ 🔍 devtree
EXPLORER ... ProfileView.tsx TS index.ts X TS cors.ts
backend > src > handlers > TS index.ts > [e] getUser
75 export const getUser = async(req: Request, res: Response) => {
    return res.status(401).json({error: error.message})
}
const [, token] = bearer.split(' ')
if(!token){
    const error = new Error('No autorizado')
    return res.status(401).json({error: error.message})
}
try{
    const result = jwt.verify(token, process.env.JWT_SECRET)
    if(typeof result === 'object' && result.id){
        console.log(result.id)
    }
} catch(error){
    res.status(500).json({error: 'Token no válido'})
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto 4000
[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto 4000
{ id: '685a04325951043f5e6b7a7d', iat: 1750731788, exp: 1766283788 }
[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto 4000
685a04325951043f5e6b7a7d

```

Enviamos y ahora solo tenemos en consola el ID.

```

File Edit Selection View Go Run ... ⏪ ⏩ ⏴ ⏵ 🔍 devtree
EXPLORER ... ProfileView.tsx TS index.ts X TS cors.ts
backend > src > handlers > TS index.ts > [e] getUser
75 export const getUser = async(req: Request, res: Response) => {
    return res.status(401).json({error: error.message})
}
const [, token] = bearer.split(' ')
if(!token){
    const error = new Error('No autorizado')
    return res.status(401).json({error: error.message})
}
try{
    const result = jwt.verify(token, process.env.JWT_SECRET)
    if(typeof result === 'object' && result.id){
        const user = await User.findById(result.id)
        //si no encuentra usuario
        if(!user){
            const error = new Error('El usuario no existe')
            return res.status(404).json({error: error.message})
        }
        res.json(user)
    }
} catch(error){
    res.status(500).json({error: 'Token no válido'})
}

```

Bien, ahora iniciamos sesión, tenemos Autenticar usuario, coloco el correo y pass le doy send obtengo un JWT, ese JWT se coloca en local storage, llevamos el usuario hacia el panel en el front (profile), entonces cuando visitemos esa página, vamos a colocar con código en Auth, en Bearer, el JWT que está en local storage, entonces le damos send y vemos que se trae la información del usuario.

Ahora, como siempre, habrá personas que quieran entrar al panel del perfil sin tener el token, y les saldrá el mensaje No autorizado.

Enviamos desde postman, y nos trae la información, incluso el pass, esta hasheado pero cual es la necesidad de retornar esa información que no es necesaria.

The screenshot shows the Postman interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'DevTree' which includes 'POST Autenticar Usuarios', 'POST Registrar Usuario', and 'GET Obtener Usuario Autentico'. The main area shows a 'GET / Obtener Usuario Autentico' request with the URL 'http://localhost:4000/user'. Under the 'Authorization' tab, it's set to 'Bearer Token' with a placeholder 'Token' and a redacted value. The 'Body' tab shows a JSON response:

```

1 {
2   "_id": "685a04325951043f5e6b7a7d",
3   "handle": "rodrigocastilla",
4   "name": "Rodrigo Castilla",
5   "email": "rodrigocastilla@gmail.com",
6   "password": "$2b$10$5Q2DRpkP5107KAeXMsK0k..fK5Sx8BTm1kgDpvqpa4cEZ1OY2RE/S",
7   "__v": 0
8 }

```

The screenshot shows the Thunder Client interface. On the left, the file tree shows 'DEVTREE' with 'src' and 'handlers' folders, and 'index.ts' is selected. The right pane shows the code for 'index.ts':

```

1 export const getUser = async(req: Request, res: Response) => {
2     return res.status(401).json({error: error.message})
3 }
4 const [, token] = bearer.split(' ')
5 if(!token){
6     const error = new Error('No autorizado')
7     return res.status(401).json({error: error.message})
8 }
9 try{
10     const result = jwt.verify(token, process.env.JWT_SECRET)
11     if(typeof result === 'object' && result.id){
12         const user = await User.findById(result.id).select('name handle email')
13         //si no encuentra usuario
14         if(user){
15             const error = new Error('El usuario no existe')
16             return res.status(404).json({error: error.message})
17         }
18         res.json(user)
19     }
20 }catch(error){
21     res.status(500).json({error: 'Token no válido'})
22 }

```

A specific line of code, `const user = await User.findById(result.id).select('name handle email')`, is highlighted with a red box.

Body Cookies Headers (8) Test Results | ⏱

{ } JSON ▾ ▷ Preview ⚡ Visualize | ▾

```

1  {
2      "_id": "685a04325951043f5e6b7a7d",
3      "handle": "rodrigocastilla",
4      "name": "Rodrigo Castilla",
5      "email": "rodrigocastilla@gmail.com"
6  }

```

“Tráeme todos menos el password”:

```

File Edit Selection View Go Run ... ← → devtree
EXPLORER ... ProfileView.ts TS index.ts x TS cors.ts
> THUNDER CLIENT: THUNDER CLIENT
DEVTREE
  backend
    > dist
    > node_modules
  src
    > config
    TS cors.ts
    TS db.ts
    handlers
      TS index.ts
    middleware
    TS validation.ts
  models
  utils
  TS auth.ts
  TS jwt.ts
  index.html
  TS index.ts
75 export const getUser = async(req: Request, res: Response) => {
76   return res.status(401).json({error: error.message})
77 }
78 const [, token] = bearer.split(' ')
79 if(!token){
80   const error = new Error('No autorizado')
81   return res.status(401).json({error: error.message})
82 }
83 try{
84   const result = jwt.verify(token, process.env.JWT_SECRET)
85   if(typeof result === 'object' && result.id){
86     const user = await User.findById(result.id).select('-password')
87     //si no encuentra usuario
88     if(user){
89       const error = new Error('El usuario no existe')
90       return res.status(404).json({error: error.message})
91     }
92     res.json(user)
93   }
94 }
95
96
97

```

Body Cookies Headers (8) Test Results | ⏱

{ } JSON ▾ ▷ Preview ⚡ Visualize | ▾

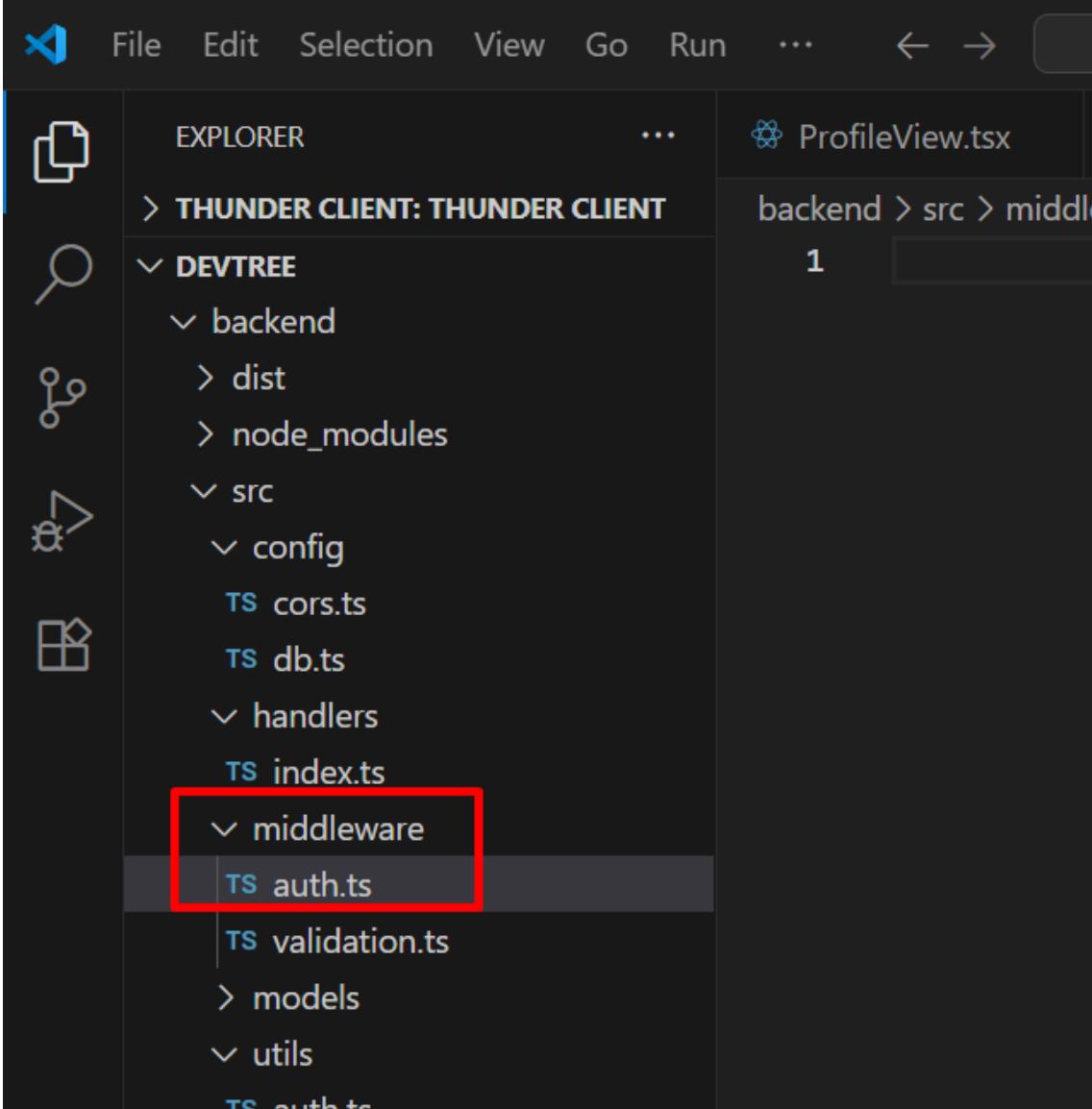
```

1  {
2      "_id": "685a04325951043f5e6b7a7d",
3      "handle": "rodrigocastilla",
4      "name": "Rodrigo Castilla",
5      "email": "rodrigocastilla@gmail.com",
6      "__v": 0
7  }

```

Esto es para el usuario, si abrimos el router solo verifica en la URL si el usuario está autenticado o no, pero que pasa si tenemos más rutas que requieren que el usuario esté autenticado. Es bastante código. Veamos como mover este código a un middleware que revise que el usuario debe estar autenticado para acceder a esta ruta.

Protegiendo el endpoint de usuario para que solo usuarios autenticados accedan:



The screenshot shows the VS Code interface with the Explorer sidebar open. The file structure is as follows:

- backend
- dist
- node_modules
- src
 - config
 - cors.ts
 - db.ts
 - handlers
 - index.ts
 - middleware
 - auth.ts
 - validation.ts
 - models
 - utils
- auth.ts

The file `auth.ts` under the `middleware` directory is highlighted with a red box in the Explorer view. Below the Explorer, the code editor shows the contents of `auth.ts`:

```
import type {Request, Response, NextFunction} from 'express'
export const authenticate = async(req: Request, res: Response, next: NextFunction) => {
}
```

Vamos a copiar el código y llevarlo al auth.ts

```

46 export const login = async(req: Request, res: Response) => {
47   res.send(token)
48 }
49
50 export const getUser = async(req: Request, res: Response) => {
51   const bearer = req.headers.authorization
52   if(!bearer){
53     const error = new Error('No autorizado')
54     return res.status(401).json({error: error.message})
55   }
56   const [, token] = bearer.split(' ')
57   if(!token){
58     const error = new Error('No autorizado')
59     return res.status(401).json({error: error.message})
60   }
61   try{
62     const result = jwt.verify(token, process.env.JWT_SECRET)
63     if(typeof result === 'object' && result.id){
64       const user = await User.findById(result.id).select('-password')
65       //si no encuentra usuario
66       if(!user){
67         const error = new Error('El usuario no existe')
68         return res.status(404).json({error: error.message})
69     }
70   }
71 }
72 }
73 }
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

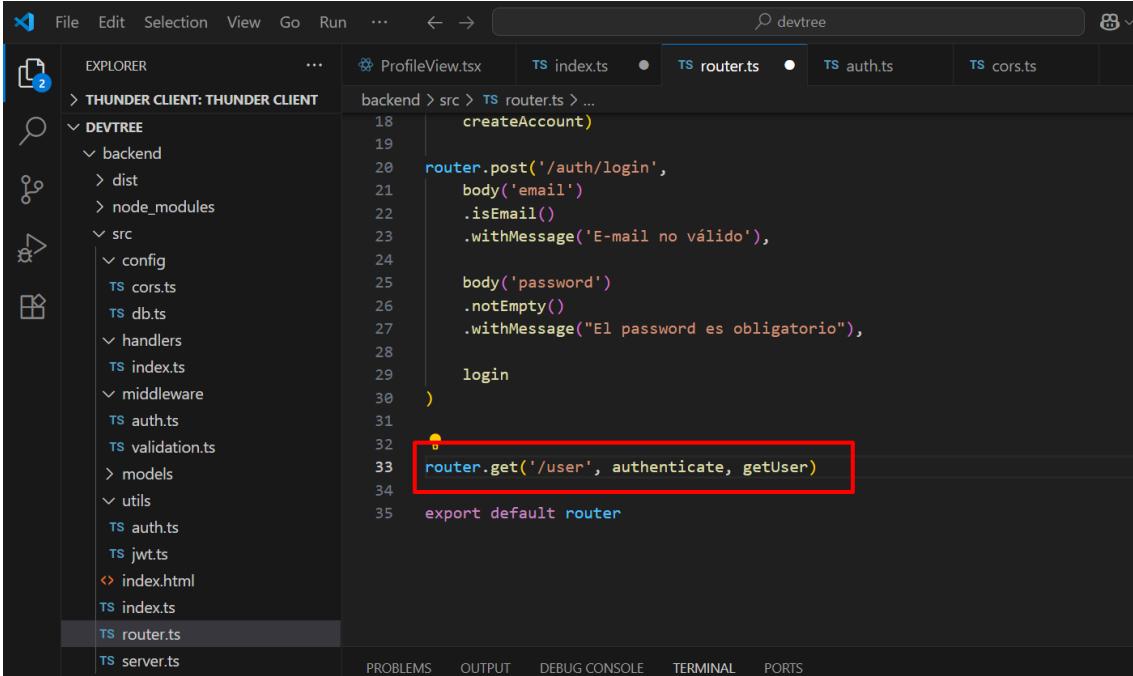
```

```

1 import type {Request, Response, NextFunction} from 'express'
2 import jwt from 'jsonwebtoken'
3 import User from '../models/User'
4
5 export const authenticate = async(req: Request, res: Response, next: NextFunction) => {
5
6   const bearer = req.headers.authorization
7   if(!bearer){
8     const error = new Error('No autorizado')
9     return res.status(401).json({error: error.message})
10  }
11  const [, token] = bearer.split(' ')
12  if(!token){
13    const error = new Error('No autorizado')
14    return res.status(401).json({error: error.message})
15  }
16  try{
17    const result = jwt.verify(token, process.env.JWT_SECRET)
18    if(typeof result === 'object' && result.id){
19      const user = await User.findById(result.id).select('-password')
20      //si no encuentra usuario
21      if(!user){
22        const error = new Error('El usuario no existe')
23        return res.status(404).json({error: error.message})
24    }
25  }
26 }
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
7
80
81
82
83
84
85
86
87
88
89
90
91
92

```

Ahora, tenemos que llamar a la función `authenticate`, y debe ser antes del `getUser`, antes de mostrar la información de quién está autenticado, requerimos revisar el JWT.



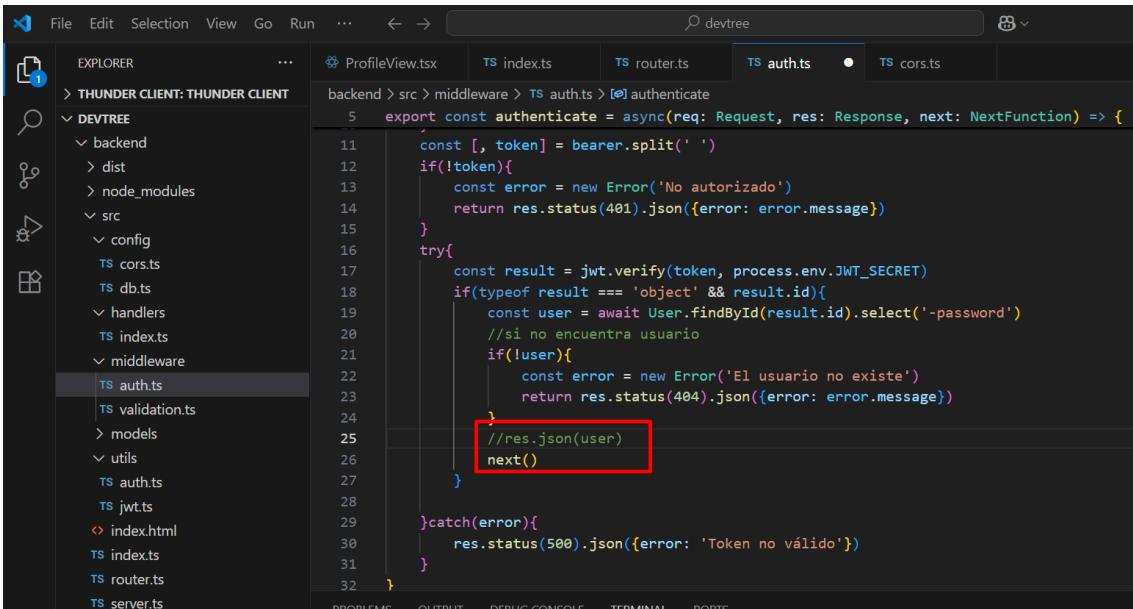
```

File Edit Selection View Go Run ... ← → devtree
EXPLORER ... ProfileView.tsx TS index.ts TS router.ts TS auth.ts TS cors.ts
> THUNDER CLIENT: THUNDER CLIENT
  < DEVTREE
    < backend
      > dist
      > node_modules
    < src
      < config
        TS cors.ts
        TS db.ts
      < handlers
        TS index.ts
      < middleware
        TS auth.ts
        TS validation.ts
      > models
      < utils
        TS auth.ts
        TS jwt.ts
      < index.html
      TS index.ts
      TS router.ts
      TS server.ts
    PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  
```

```

backend > src > TS router.ts > ...
18   createAccount)
19
20   router.post('/auth/login',
21     body('email')
22     .isEmail()
23     .withMessage('E-mail no válido'),
24
25     body('password')
26     .notEmpty()
27     .withMessage("El password es obligatorio"),
28
29   login
30 )
31
32
33 router.get('/user', authenticate, getUser)
34
35 export default router
  
```

Como se obtiene una respuesta ya no entra al método getUser, entonces agregamos un next.



```

File Edit Selection View Go Run ... ← → devtree
EXPLORER ... ProfileView.tsx TS index.ts TS router.ts TS auth.ts TS cors.ts
> THUNDER CLIENT: THUNDER CLIENT
  < DEVTREE
    < backend
      > dist
      > node_modules
    < src
      < config
        TS cors.ts
        TS db.ts
      < handlers
        TS index.ts
      < middleware
        TS auth.ts
        TS validation.ts
      > models
      < utils
        TS auth.ts
        TS jwt.ts
      < index.html
      TS index.ts
      TS router.ts
      TS server.ts
    PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  
```

```

backend > src > middleware > TS auth.ts > authenticate
5   export const authenticate = async(req: Request, res: Response, next: NextFunction) => {
11     const [, token] = bearer.split(' ')
12     if(!token){
13       const error = new Error('No autorizado')
14       return res.status(401).json({error: error.message})
15     }
16     try{
17       const result = jwt.verify(token, process.env.JWT_SECRET)
18       if(typeof result === 'object' && result.id){
19         const user = await User.findById(result.id).select('-password')
20         //si no encuentra usuario
21         if(!user){
22           const error = new Error('El usuario no existe')
23           return res.status(404).json({error: error.message})
24         }
25       }
26       //res.json(user)
27       next()
28     }
29     }catch(error){
30       res.status(500).json({error: 'Token no válido'})
31     }
32   }
  
```

```

File Edit Selection View Go Run ...
ProfileView.tsx TS index.ts x TS router.ts TS auth.ts TS cors.ts
EXPLORER ...
> THUNDER CLIENT: THUNDER CLIENT
  < DEVTREE
    < backend
      > dist
      > node_modules
    < src
      < config
        TS cors.ts
        TS db.ts
      < handlers
        TS index.ts
      < middleware
        TS auth.ts
        TS validation.ts
      > models
      < utils
        TS auth.ts
        TS jwt.ts
      < index.html
    
```

```

46  export const login = async(req: Request, res:Response) => {
47
48    if(!isPasswordCorrect){
49      const error = new Error('Password incorrecto')
50      return res.status(401).json({error: error.message})
51      //401: porque no está autorizado a acceder a este recurso
52    }
53
54    //Hasta aquí tenemos un usuario que ingresó correctamente, entonces:
55
56    const token = generateJWT({id: user._id})
57    res.send(token)
58  }
59
60  export const getUser = async(req: Request, res: Response) => {
61    console.log('Desde getUser')
62  }
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
  
```

Enviamos desde postman

POST Autenticar Usuarios • POST Registrar Usuario • GET Obtener Usuario Autentico • +

Save Share

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Response

```

[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts --api`
Servidor funcionando en el puerto 4000
Desde getUser
  
```

Ahora, algo debe detener la ejecución. Entonces, antes de irnos al siguiente middleware podemos colocarlo en el request del usuario:

```

File Edit Selection View Go Run ... < > devtree
EXPLORER ... ProfileView.tsx TS index.ts TS auth.ts ● TS router.ts TS cors.ts
backend > src > middleware > TS auth.ts > authenticate
5   export const authenticate = async(req: Request, res: Response, next: NextFunction) => {
12     if(!token){
13       const error = new Error('No autorizado')
14       return res.status(401).json({error: error.message})
15     }
16     try{
17       const result = jwt.verify(token, process.env.JWT_SECRET)
18       if(typeof result === 'object' && result.id){
19         const user = await User.findById(result.id).select('-password')
20         //si no encuentra usuario
21         if(!user){
22           const error = new Error('El usuario no existe')
23           return res.status(404).json({error: error.message})
24         }
25         //res.json(user)
26         req.user = user
27         next()
28       }
29     }catch(error){
30       res.status(500).json({error: 'Token no válido'})
31     }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

File Edit Selection View Go Run ... < > devtree
EXPLORER ... ProfileView.tsx TS index.ts ✘ TS auth.ts TS router.ts TS cors.ts
backend > src > handlers > TS index.ts > getUser
46   export const login = async(req: Request, res: Response) => {
47     if(!isPasswordCorrect){
48       const error = new Error('Password incorrecto')
49       return res.status(401).json({error: error.message})
50       //401: porque no está autorizado a acceder a este recurso
51     }
52     //Hasta aquí tenemos un usuario que ingresó correctamente, entonces:
53     const token = generateJWT({id: user._id})
54     res.send(token)
55   }
56   export const getUser = async(req: Request, res: Response) => {
57     res.json(req.user)
58   }

```

Guardamos cambios y el app se crashea.

Este request es algo interno de express y estamos agregándole una propiedad que no conoce, pero podemos agregar:

The screenshot shows a developer's environment with two main windows. The top window is a code editor in VS Code, displaying a file named 'TS auth.ts'. A red box highlights a section of the code where a global interface 'IUser' is declared:

```

1 import type {Request, Response, NextFunction} from 'express'
2 import jwt from 'jsonwebtoken'
3 import User, { IUser } from '../models/User'
4
5 declare global{
6   namespace Express{
7     interface Request{
8       | user?: IUser
9     }
10  }
11}
12
13 export const authenticate = async(req: Request, res: Response, next: NextFunction) => {
14  const bearer = req.headers.authorization
15  if(!bearer){
16    const error = new Error('No autorizado')
17    return res.status(401).json({error: error.message})
18  }
19  const [, token] = bearer.split(' ')
20  if(!token){
21    const error = new Error('No autorizado')

```

The bottom window is a browser-based API testing tool called Postman. It shows a 'My Workspace' collection with several requests. One request is selected: 'GET Obtener Usuario Autenticado'. The 'Headers' tab is open, showing 'Authorization' set to 'Bearer Token' with a placeholder '.....'. The 'Body' tab shows a JSON response:

```

1 {
2   "_id": "685a04325951043f5e6b7a7d",
3   "handle": "rodrigocastilla",
4   "name": "Rodrigo Castilla",
5   "email": "rodrigocastilla@gmail.com",
6   "__v": 0

```

Ahora, queremos restringir el acceso a algunas rutas, veamos.

Vamos a trabajar en el frontend. Vamos a escribir un código que revise que el usuario tenga un JWT que sea válido y tenga acceso a las URL. Muchas veces, cuando queremos enviar una petición tan pronto como cargue el componente hacemos la consulta en un UseEffect, pero las consultas no son tan rápidas. Existe una alternativa para obtener datos desde una Api, así que veamos que es ReactQuery.

¿Qué es React Query?

- React Query o también TanStack Query es una librería para obtener datos del servidor.
- Sus ventajas principales son que obtiene los datos de forma optimizada y rápida; además cachea las consultas, sincroniza / actualiza los datos del servidor de forma muy simple.
- Se puede utilizar con y sobre Fetch API o Axios.
- En este caso usaremos Axios.

Términos en React Query

React Query introduce una gran cantidad de conceptos nuevos; pero hay 2 que son los más importantes:

- Queries: Se utilizan para obtener datos de un servidor o una API (GET)
- Mutations: Se utilizan para crear / actualizar / eliminar datos en el servidor (POST, PUT, PATCH, DELETE)
- Por ahora, nuestro proyecto tenemos POST – Registro y autenticación.

Queries (useQuery)

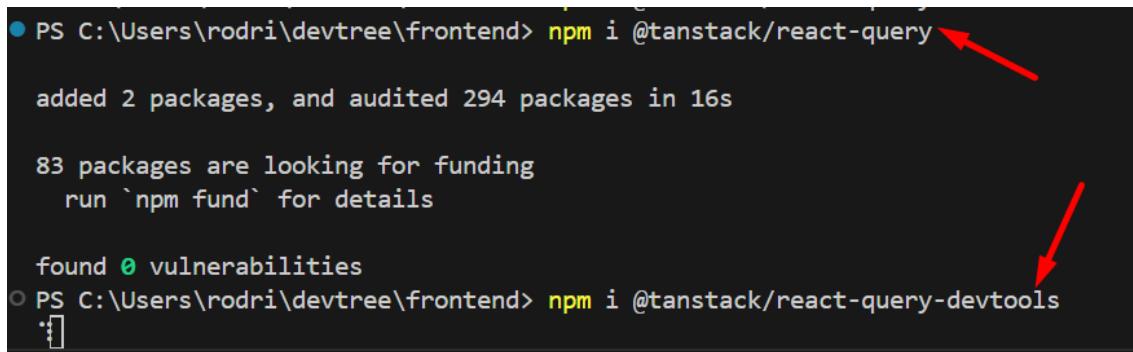
Se utilizan para obtener datos de un servidor o una API (GET)

Mutations (useMutation)

Se utilizan para crear / actualizar / eliminar datos en el servidor (POST, PUT, PATCH, DELETE)

Primeros pasos con React Query.

tanstack.com/query/latest



```
● PS C:\Users\rodri\devtree\frontend> npm i @tanstack/react-query
added 2 packages, and audited 294 packages in 16s
83 packages are looking for funding
  run `npm fund` for details

  found 0 vulnerabilities
○ PS C:\Users\rodri\devtree\frontend> npm i @tanstack/react-query-devtools
:[]
```

The screenshot shows a Windows Command Prompt (PS) window. The first command, 'npm i @tanstack/react-query', is highlighted with a red arrow pointing to the package name. The second command, 'npm i @tanstack/react-query-devtools', is also highlighted with a red arrow pointing to the package name.

El segundo nos va a permitir revisar los resultados de las consultas.

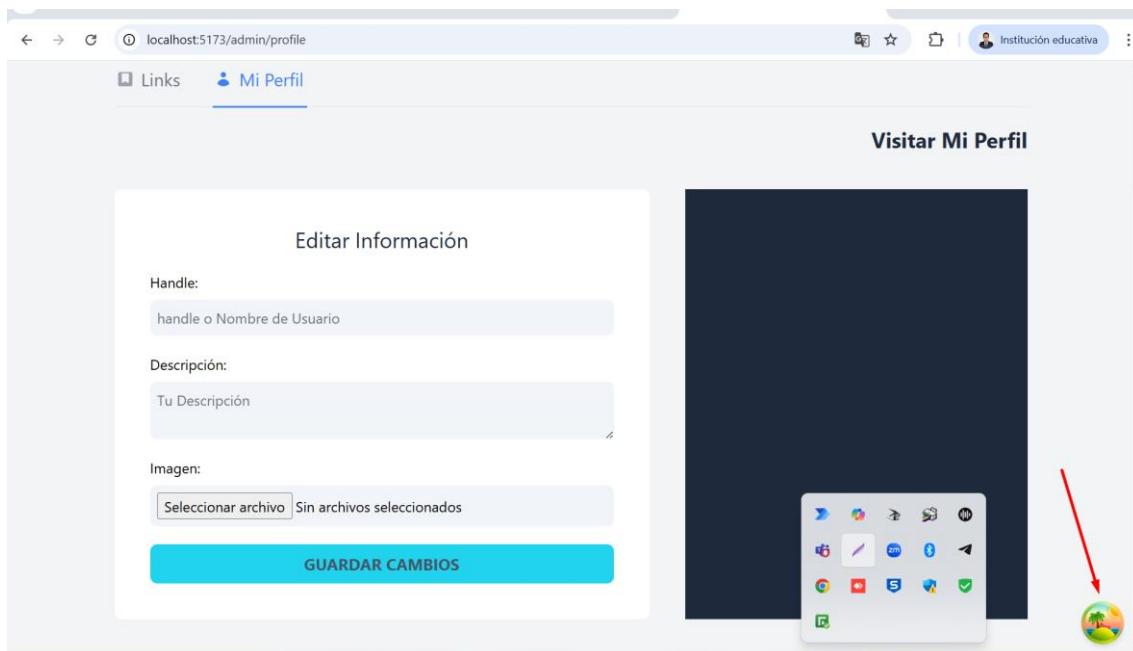
```

File Edit Selection View Go Run ... ← → ⌂ devtree
EXPLORER main.tsx •
THUNDER CLIENT: THUNDER CLIENT
DEVTREE
frontend > src > main.tsx > ...
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
4 import { ReactQueryDevtools } from '@tanstack/react-query-devtools'
5 import Router from './router'
6 import './index.css'
7
8 const queryClient = new QueryClient()
9
10 ReactDOM.createRoot(document.getElementById('root')!).render(
11   <React.StrictMode>
12     <QueryClientProvider client={queryClient}>
13       <Router/>
14       <ReactQueryDevtools/>
15     </QueryClientProvider>
16   </React.StrictMode>,
17 )
18

```

Colocamos el router dentro del queryclientprovider, de esta forma va a estar habilitado react query de forma global en todos los componentes de la app y del router.

De esta forma está instalado en toda la app, reactquery.



Diferentes herramientas. Como hacer la consulta desde reactquery, veamos.

Utilizando el Hook useQuery

En este caso es una petición de tipo Get, y como vimos, cuando es Get utilizamos el Hook de UseQuery.

```
main.tsx  AppLayout.tsx 2

frontend > src > layouts > AppLayout.tsx > AppLayout
  1 import { Link, Outlet } from "react-router-dom";
  2 import { Toaster } from "sonner";
  3 import { useQuery } from '@tanstack/react-query';
  4 import NavigationTabs from "../components/NavigationTabs";
  5
  6 export default function AppLayout() {
  7
  8   const { data, error } = useQuery({
  9     //pasarle algunas configuraciones
 10     queryFn: //qué función va a hacer la consulta hacia nuestra api
 11   })
 12
 13   return (
 14     <>
 15       <header className="bg-slate-800 py-5">
 16         <div className="mx-auto max-w-5xl flex flex-col md:flex-row items-center md:justify-between">
 17           <div className="w-full p-5 lg:p-0 md:w-1/3">
 18             
 19           </div>
 20           <div className="d:w-1/3 md:flex md:justify-end">
```

The screenshot shows the VS Code interface with the Devtree extension installed. The Explorer sidebar on the left displays a tree view of the project structure under 'THUNDER CLIENT: THUNDER CLIENT'. A red box highlights the 'api' folder within the 'src' directory. The main editor area shows the content of 'DevTreeAPI.ts'. The status bar at the bottom indicates the file is 2 lines long.

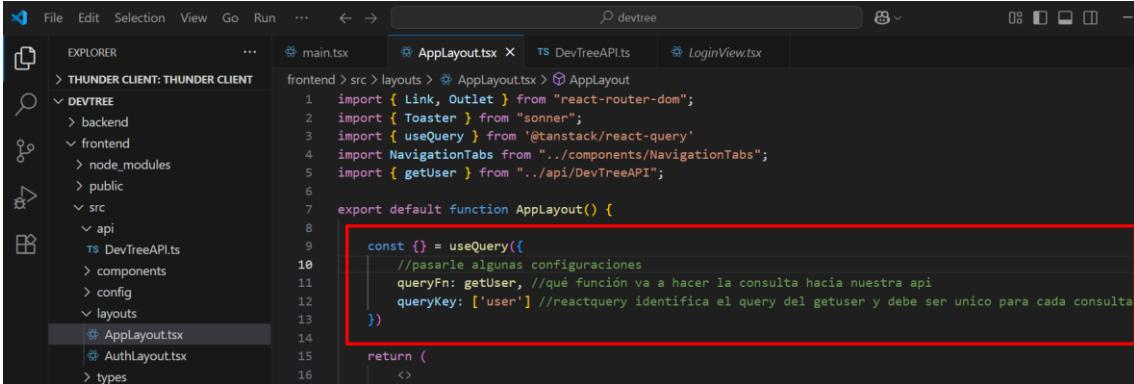
```
1 export async function getUser(){
2     try{
3
4         }catch(error){
5
6     }
7 }
```

Copiar el código del LoginView.

```
import { isAxiosError } from "axios"
import api from "../config/axios"

export async function getUser(){
    try{
        //tenemos que hacer la petición hacia la URL
        const {data} = await api(`user`)
        console.log(data)
        localStorage.setItem('AUTH_TOKEN', data)
    }catch(error){
        if(isAxiosError(error) && error.response){
            console.log(error.response.data.error)
        }
    }
}
```

Vamos A volver al AppLayout y en queryfunction hay que especificar el getUser.

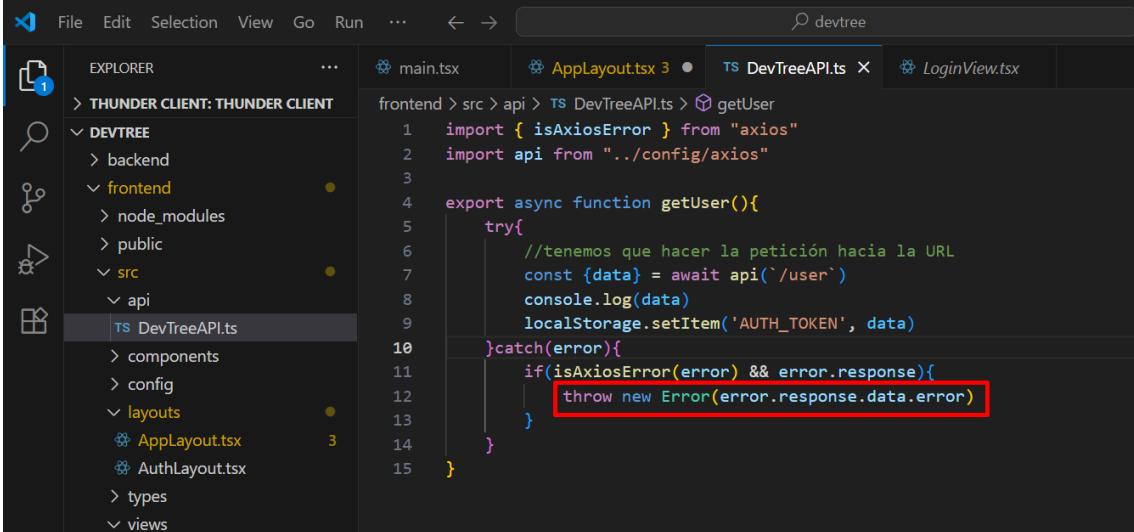


```

File Edit Selection View Go Run ... < > devtree
EXPLORER
> THUNDER CLIENT: THUNDER CLIENT
  > DEVTREE
    > backend
    > frontend
      > node_modules
      > public
      > src
        > api
          DevTreeAPI.ts
        > components
        > config
        > layouts
          AppLayout.tsx
          AuthLayout.tsx
        > types
        > views
main.tsx AppLayouttsx TS DevTreeAPIts LoginView.tsx
frontend > src > layouts > AppLayouttsx > AppLayout
1 import { Link, Outlet } from "react-router-dom";
2 import { Toaster } from "sonner";
3 import { useQuery } from '@tanstack/react-query';
4 import NavigationTabs from "../components/NavigationTabs";
5 import { getUser } from "../api/DevTreeAPI";
6
7 export default function AppLayout() {
8
9   const {} = useQuery({
10     //pasarle algunas configuraciones
11     queryFn: getUser, //qué función va a hacer la consulta hacia nuestra api
12     queryKey: ['user'] //reactquery identifica el query del getUser y debe ser unico para cada consulta
13   })
14
15   return (
16     <>

```

Propiedades de UseQuery: data, isLoading, error, isError

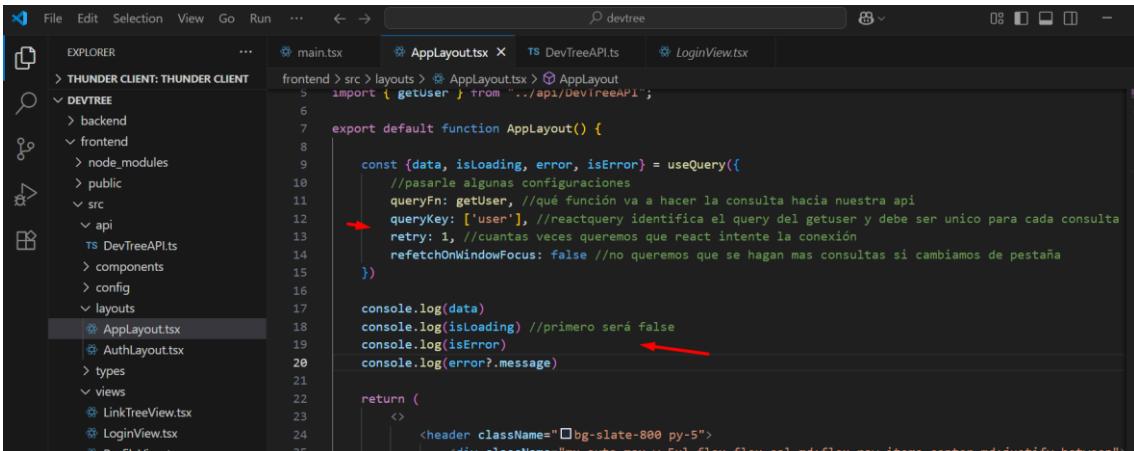


```

File Edit Selection View Go Run ... < > devtree
EXPLORER
> THUNDER CLIENT: THUNDER CLIENT
  > DEVTREE
    > backend
    > frontend
      > node_modules
      > public
      > src
        > api
          DevTreeAPI.ts
        > components
        > config
        > layouts
          AppLayout.tsx
          AuthLayout.tsx
        > types
        > views
main.tsx AppLayouttsx 3 TS DevTreeAPI.ts LoginView.tsx
frontend > src > api > DevTreeAPI.ts > getUser
1 import { isAxiosError } from "axios"
2 import api from "../config/axios"
3
4 export async function getUser(){
5   try{
6     //tenemos que hacer la petición hacia la URL
7     const {data} = await api(`user`)
8     console.log(data)
9     localStorage.setItem('AUTH_TOKEN', data)
10  }catch(error){
11    if(isAxiosError(error) && error.response){
12      throw new Error(error.response.data.error)
13    }
14  }
15 }

```

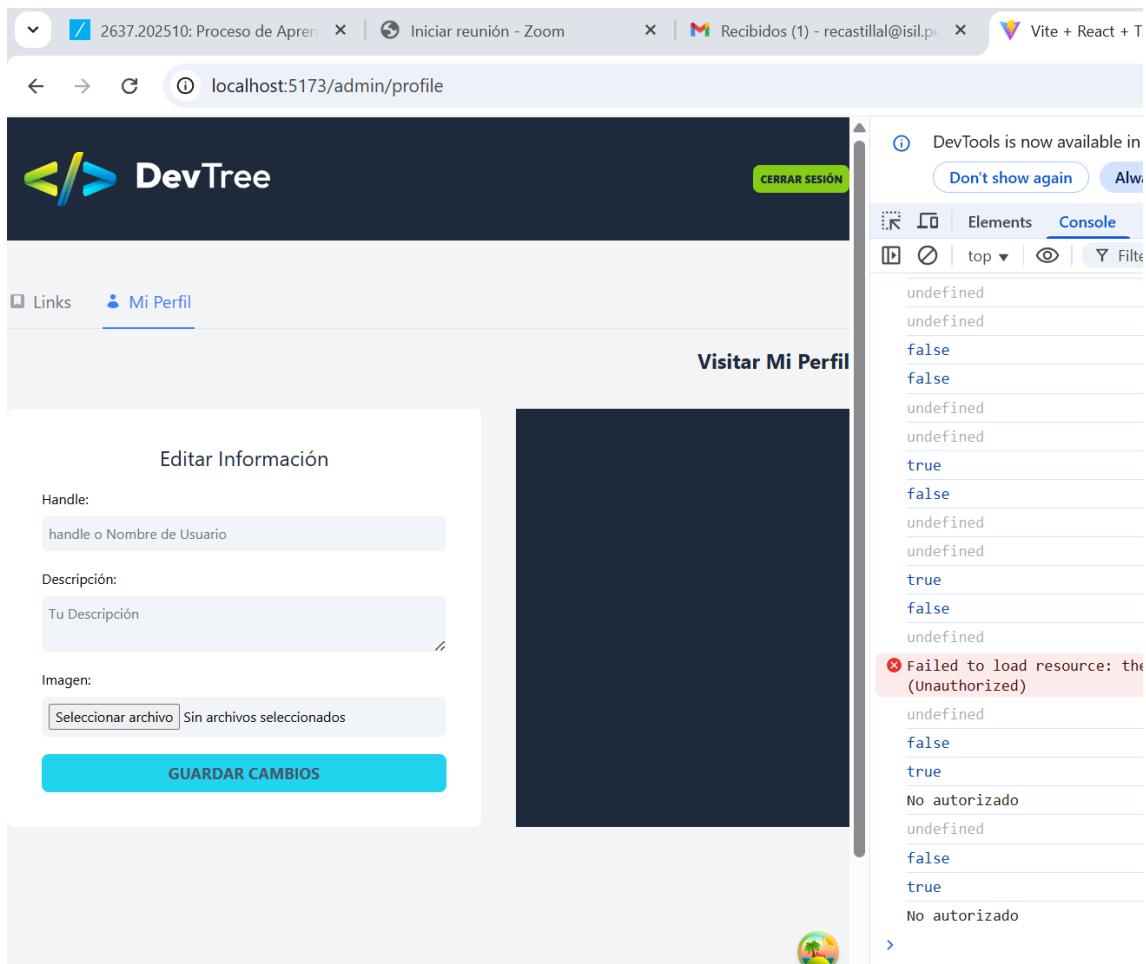
Hacemos eso porque en el applayout tenemos error, isError.



```

File Edit Selection View Go Run ... < > devtree
EXPLORER
> THUNDER CLIENT: THUNDER CLIENT
  > DEVTREE
    > backend
    > frontend
      > node_modules
      > public
      > src
        > api
          DevTreeAPI.ts
        > components
        > config
        > layouts
          AppLayout.tsx
          AuthLayout.tsx
        > types
        > views
          LinkTreeView.tsx
          LoginView.tsx
          ProfileFooter.tsx
main.tsx AppLayouttsx TS DevTreeAPIts LoginView.tsx
frontend > src > layouts > AppLayouttsx > AppLayout
5 import { getUser } from "../api/DevTreeAPI";
6
7 export default function AppLayout() {
8
9   const {data, isLoading, error, isError} = useQuery({
10     //pasarle algunas configuraciones
11     queryFn: getUser, //qué función va a hacer la consulta hacia nuestra api
12     queryKey: ['user'], //reactquery identifica el query del getUser y debe ser unico para cada consulta
13     retry: 1, //cuantas veces queremos que react intente la conexión
14     refetchOnWindowFocus: false //no queremos que se hagan mas consultas si cambiamos de pestaña
15   })
16
17   console.log(data)
18   console.log(isLoading) //primero será false
19   console.log(isError)
20   console.log(error?.message)
21
22   return (
23     <>
24       <header className="bg-slate-800 py-5">
25         <div className="mx-auto max-w-5xl flex flex-col md:flex-row items-center justify-between">

```



Udefined: data

True: isLoading

Primero es false, luego true: isError

Error: No autorizado

Veamos cómo solucionar el problema del error “no autorizado”, y tener al usuario autenticado en nuestra aplicación.

Enviando la petición con el JWT en ReactQuery

En auth.ts tenemos un no autorizado cuando no enviamos un bearer o cuando no enviamos un token.

Sabemos que tenemos un token guardado en local storage, entonces podemos enviar ese JWT vía header. En DevTreeApi.ts

```

File Edit Selection View Go Run ... ⏪ ⏹ devtree
EXPLORER main.tsx AppLayout.tsx TS DevTreeAPI.ts x LoginView.tsx
frontend > src > api > TS DevTreeAPI.ts > ⚡ getUser
1 import { isAxiosError } from "axios"
2 import api from "../config/axios"
3
4 export async function getUser(){
5   const token = localStorage.getItem('AUTH_TOKEN')
6   try{
7     //tenemos que hacer la petición hacia la URL
8     const {data} = await api('/user', {
9       headers: {
10         Authorization: `Bearer: ${token}` //de esta manera se envía Bearer con el JWT
11       }
12     })
13   return data
14   //console.log(data)
15   //localStorage.setItem('AUTH_TOKEN', data)
16 }catch(error){
17   if(isAxiosError(error) && error.response){
18     throw new Error(error.response.data.error)
19   }
20 }

```

En consola ya aparece la información del usuario.

localhost:5173/admin/profile

DevTools is now available in Spanish

Console

```

react-dom_client.js?v=5dc04d2717987
undefined
true
false
undefined
undefined
true
false
undefined
[{"id": "685004325921043f5e6b7a7d", "handle": "rodrigocastilla", "name": "Rodrigo castilla", "email": "rodri...@gmail.com", "__v": 0}
false
false
undefined
AppLayout.tsx:17
AppLayout.tsx:18
AppLayout.tsx:19
AppLayout.tsx:20
AppLayout.tsx:17
AppLayout.tsx:18
AppLayout.tsx:19
AppLayout.tsx:20
AppLayout.tsx:17
AppLayout.tsx:18
AppLayout.tsx:19
AppLayout.tsx:20
AppLayout.tsx:17
AppLayout.tsx:18
AppLayout.tsx:19
AppLayout.tsx:20
AppLayout.tsx:17

```

La parte sombreada la vamos a repetir siempre, y vamos a mejorar eso con algo llamado Interceptors.

```

File Edit Selection View Go Run ... ⏪ ⏹ devtree
EXPLORADOR Obtener Usuario Autenticado... ProfileView.tsx AppLayout.tsx TS DevTreeAPI.ts x TS auth.ts
frontend > src > api > TS DevTreeAPI.ts > ⚡ getUser
1 import { isAxiosError } from 'axios'
2 import api from '../config/axios'
3
4 export async function getUser() {
5   const token = localStorage.getItem('AUTH_TOKEN')
6   try {
7     const { data } = await api('/user', [
8       headers: {
9         Authorization: `Bearer: ${token}`
10      }
11    ])
12   return data
13 } catch (error) {
14   if (isAxiosError(error) && error.response) {
15     throw new Error(error.response.data.error)
16   }
17 }

```

Añadiendo el JWT en un interceptor de Axios

Nos va a ayudar en dos cosas, primero, si tengo más funciones que requieran que el usuario esté autenticado pues requiero enviar el mismo JWT, lo cual va a llevar a funciones más grandes y repetición de código. Con un interceptor podemos colocar el JWT y que se envíe en automático en caso sea necesario, y si no, se ignore.

```

File Edit Selection View Go Run ... < > devtree
EXPLORER ... main.tsx AppLayout.tsx TS DevTreeAPI.ts ✘ TS axios.ts
> THUNDER CLIENT: THUNDER CLIENT
  < DEVTREE
    > backend
    < frontend
      > node_modules
      > public
      < src
        < api
          TS DevTreeAPI.ts
            > components
            < config
              TS axios.ts
            < layouts
              AppLayout.tsx
              AuthLayout.tsx
            > types
    > types
TS DevTreeAPI.ts
  > src > api > TS DevTreeAPI.ts > getUser
  1 import { isAxiosError } from "axios"
  2 import api from "../config/axios"
  3
  4 export async function getUser(){
  5   const token = localStorage.getItem('AUTH_TOKEN')
  6   try{
  7     //tenemos que hacer la petición hacia la URL
  8     const {data} = await api('/user', {
  9       headers: {
  10         Authorization: `Bearer: ${token}` //de esta manera se envía Bearer con el JWT
  11       }
  12     })
  13     return data
  14     //console.log(data)
  15     //localStorage.setItem('AUTH_TOKEN', data)
  16   }catch(error){
  17     if(isAxiosError(error) && error.response){

```

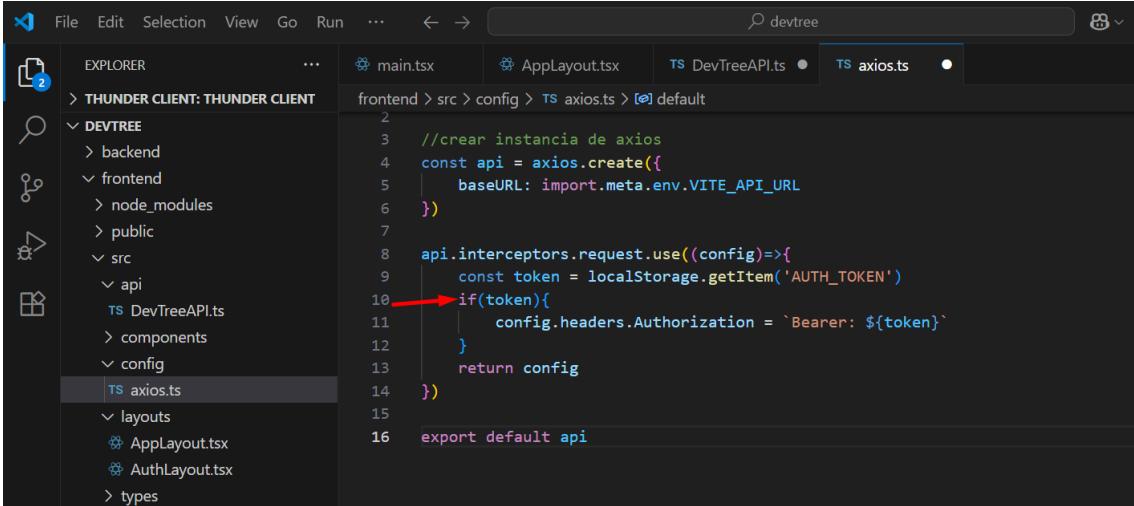
Cortar y llevarlo a axios.

```

File Edit Selection View Go Run ... < > devtree
EXPLORER ... main.tsx AppLayout.tsx TS DevTreeAPI.ts ✘ TS axios.ts
> THUNDER CLIENT: THUNDER CLIENT
  < DEVTREE
    > backend
    < frontend
      > node_modules
      > public
      < src
        < api
          TS DevTreeAPI.ts
        > components
        < config
          TS axios.ts
        < layouts
          AppLayout.tsx
          AuthLayout.tsx
        > types
        < views
          LinkTreeView.tsx
TS axios.ts
  > src > config > TS axios.ts > ...
  1 import axios from 'axios'
  2
  3 //crear instancia de axios
  4 const api = axios.create({
  5   baseURL: import.meta.env.VITE_API_URL
  6 })
  7
  8 api.interceptors.request.use((config)=>{
  9   const token = localStorage.getItem('AUTH_TOKEN')
 10
 11   config.headers.Authorization = `Bearer: ${token}`
 12
 13   return config
 14 })
 15
 16 export default api

```

Qué pasa con aquellas personas que recién están ingresando a la aplicación y no tienen un token, eso hará que se envié algo que no existe, y en JavaScript cuando tienes un error en el código todo lo demás deja de funcionar.



```

1 //crear instancia de axios
2 const api = axios.create({
3   baseURL: import.meta.env.VITE_API_URL
4 })
5
6
7 api.interceptors.request.use((config)=>{
8   const token = localStorage.getItem('AUTH_TOKEN')
9   if(token){
10     config.headers.Authorization = `Bearer: ${token}`
11   }
12   return config
13 }
14 )
15
16 export default api

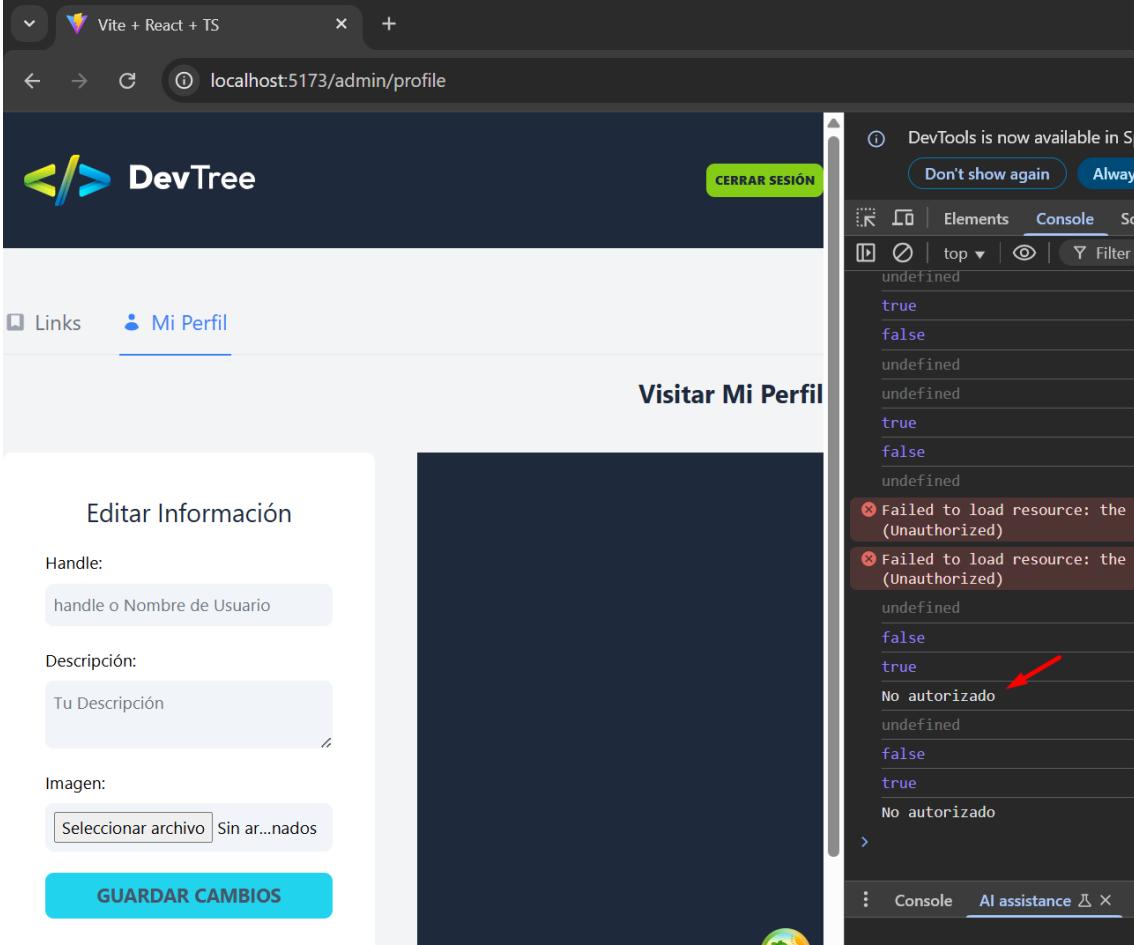
```

De esta forma si tengo más peticiones, el interceptor se encarga de enviarlo cuando se requiera.

Ahora, si un usuario no tiene un jwt no tiene por qué ver la página.

Proteger el Panel de Administración si un usuario no está autenticado

Si abrimos un incógnito, nos aparece “no autorizado”.



The browser window displays a login form for 'DevTree' with fields for 'Handle' and 'Descripción'. The right side of the screen shows the browser's developer tools DevTools open, specifically the Console tab. The console output shows several error messages related to failed resource loads, all labeled '(Unauthorized)'. A red arrow points to the last message in the list, which reads 'No autorizado'.

```

File Edit Selection View Go Run ...
main.tsx AppLayout.tsx DevTreeAPI.ts axios.ts
frontend > src > layouts > AppLayout.tsx
1 import { Link, Outlet } from "react-router-dom";
2 import { Toaster } from "sonner";
3 import { useQuery } from '@tanstack/react-query'
4 import { Navigate } from 'react-router-dom'
5 import NavigationTabs from "../components/NavigationTabs";
6 import { getUser } from "../api/DevTreeAPI";
7
8 export default function AppLayout() {
9
10   const {data, isLoading, isError} = useQuery({
11     //pasarle algunas configuraciones
12     queryFn: getUser, //qué función va a hacer la consulta hacia nuestra api
13     queryKey: ['user'], //reactquery identifica el query del getuser y debe ser unico para cada consulta
14     retry: 1, //cuantas veces queremos que react intente la conexión
15     refetchOnWindowFocus: false //no queremos que se hagan mas consultas si cambiamos de pestaña
16   })
17
18   if(isLoading) return 'Cargando...'
19   if(isError){
20     return <Navigate to={'/auth/login'}>
21   }
22
23   //console.log(data)
24   //console.log(isLoading) //primero será false
25   //console.log(isError)
26   //console.log(error?.message)

```

Probamos desde el incógnito y validamos que falla y nos lleva hacia iniciar sesión.

Lo que hemos hecho también protege LinkTreeView.

Obteniendo la información del usuario y mostrándola en el componente

Abrimos getUser y le colocamos vía generic, que la respuesta de la api va a retornarnos un usuario.

```

File Edit Selection View Go Run ...
main.tsx AppLayout.tsx DevTreeAPI.ts axios.ts
frontend > src > api > DevTreeAPI.ts > getUser
1 import { isAxiosError } from "axios"
2 import api from "../config/axios"
3 import type { User } from "../types" -----
4
5 export async function getUser(){
6
7   try{
8     //tenemos que hacer la petición hacia la URL
9     const {data} = await api<User>('/user', )
-----
10    return data
11    //console.log(data)
12    //localStorage.setItem('AUTH_TOKEN', data)
13  }catch(error){
14    if(isAxiosError(error) && error.response){
15      throw new Error(error.response.data.error)
16    }
17  }
18

```

Y de esa forma ese “data” ya es de tipo User.

Verificamos que “data” es de tipo de User, pero También es undefined.

El objeto real tiene un ID, pero nuestro type no lo tiene. Podemos agregar el ID en el type.

Otra forma es usar Zod, que permite validar las respuestas de las Apis, es decir, yo le digo en DevTreeApi, que debe ser un usuario y Zod escanea el data, y valida si es o no un usuario. Les dejo de tarea.

The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows the project structure under "THUNDER CLIENT: THUNDER CLIENT".
 - DEVTREE
 - > backend
 - < frontend
 - > node_modules
 - > public
 - < src
 - < api
 - TS DevTreeAPI.ts
 - < components
 - < config
 - TS axios.ts
 - < layouts
 - TS AppLayout.tsx (highlighted)
 - TS AuthLayout.tsx
 - < types
 - < views
 - TS LinkTreeView.tsx
- Code Editor:** The file "AppLayout.tsx" is open. A red box highlights the line of code: `if(data) return (`. The code block contains:

```
frontend > src > layouts > AppLayout.tsx > AppLayout
8   export default function AppLayout() {
25     //console.log(isError)
26     //console.log(error?.message)
27
28     if(data) return (
29       <>
30         <header className="bg-slate-800 py-5">
31           <div className="mx-auto max-w-5xl flex flex-col w-full p-5 lg:p-0 md:w-1/3">
32             <div className="w-full p-5 lg:p-0 md:w-1/3">
33               
34             </div>
35             <div className="md:w-1/3 md:flex md:justify-between px-4 py-2">
36               <button
37                 className="bg-lime-500 p-2 text-white rounded-md font-medium"
38                 onClick={() => {}}>
39                 Cerrar Sesión
40               </button>
41             </div>
42           </div>
43         </header>
44       </div>
45     )
46   )
47 }
```

The screenshot shows a web browser window with the following details:

- URL:** localhost:5173/admin/profile
- Header:** DevTree
- Buttons:** CERRAR SESIÓN
- Navigation:** Links, Mi Perfil (selected)
- Content:**
 - Visitar Mi Perfil: /rodrigocastilla** (highlighted with a red box)
 - Editar Información**
 - Handle:** handle o Nombre de Usuario
 - Descripción:** Tu Descripción
 - A large dark rectangular placeholder for a profile picture.

Más adelante estaremos creando la sección de los enlaces. Se recomienda es cortar todo ese código, todo ese fragmento y lo llevamos a su propio componente.

```

File Edit Selection View Go Run ... < > devtree
EXPLORER > THUNDER CLIENT: THUNDER CLIENT
  > DEVTREE
    > backend
    > frontend
      > node_modules
      > public
      > src
        > api DevTreeAPI.ts
        > components DevTree.tsx
          DevTree.tsx 7
        > config axios.ts
        > layouts AppLayout.tsx 4
          AppLayout.tsx
        > AuthLayout.tsx
      > layouts
      > types
      > views
      > LinkTreeView.tsx
      > LoginView.tsx
      > ProfileView.tsx
      > RegisterView.tsx
      # index.css
      > main.tsx
frontend > src > components > DevTree.tsx > DevTree
1 import React from 'react'
2
3 export default function DevTree(){
4   return(
5     <>
6       <header className="bg-slate-800 py-5">
7         <div className="mx-auto max-w-5xl flex flex-col md:flex-row items-center md:justify-between">
8           <div className="w-full p-5 lg:p-0 md:w-1/3">
9             
10            </div>
11            <div className="md:w-1/3 md:flex md:justify-end">
12              <button
13                className="bg-lime-500 p-2 text-slate-800 uppercase font-black text-xs rounded-md"
14                onClick={() => {}}
15              >
16                Cerrar Sesión
17              </button>
18            </div>
19          </div>
20        </header>
21      <>
22    </>
23  </>
24</>
25</>
26</>
27</>
28</>
29</>
30</>

```

PEGAMOS DESDE APPLAYOUT

```

File Edit Selection View Go Run ... < > devtree
EXPLORER > THUNDER CLIENT: THUNDER CLIENT
  > DEVTREE
    > backend
    > frontend
      > node_modules
      > public
      > src
        > api DevTreeAPI.ts
        > components DevTree.tsx
          DevTree.tsx 7
        > config axios.ts
        > layouts AppLayout.tsx 3
          AppLayout.tsx
        > AuthLayout.tsx
      > layouts
      > types
      > views
      > LinkTreeView.tsx
      > LoginView.tsx
      > ProfileView.tsx
      > RegisterView.tsx
      # index.css
      > main.tsx
frontend > src > layouts > AppLayout.tsx > AppLayout
6 import { getUser } from "../api/DevTreeAPI";
7 import DevTree from "../components/DevTree";
8
9 export default function AppLayout() {
10
11   const {data, isLoading, isError} = useQuery({
12     //pasarle algunas configuraciones
13     queryFn: getUser, //qué función va a hacer la consulta hace
14     queryKey: ['user'], //reactquery identifica el query del g
15     retry: 1, //cuantas veces queremos que react intente la co
16     refetchOnWindowFocus: false //no queremos que se hagan mas
17   })
18
19   if(isLoading) return 'Cargando...'
20   if(isError){
21     return <Navigate to={'/auth/login'} />
22   }
23
24   //console.log(data)
25   //console.log(isLoading) //primero será false
26   //console.log(isError)
27   //console.log(error?.message)
28
29   if(data) return <DevTree/>
30 }

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS
11:41:06 [vite] (client) hmr update /src/layouts/AppLayout.tsx, /src/index.c

Traemos las importaciones de AppLayout al DevTree.tsx

```

File Edit Selection View Go Run ... < > devtree
EXPLORER > THUNDER CLIENT: THUNDER CLIENT
  > DEVTREE
    > backend
    > frontend
      > node_modules
      > public
      > src
        > api DevTreeAPI.ts
        > components DevTree.tsx
          DevTree.tsx 2
        > config axios.ts
      > layouts
      > types
      > views
      > LinkTreeView.tsx
      > LoginView.tsx
      > ProfileView.tsx
      > RegisterView.tsx
      # index.css
      > main.tsx
frontend > src > components > DevTree.tsx > DevTree
1 import React from 'react'
2 import NavigationTabs from "../components/NavigationTabs";
3 import { Link, Outlet } from "react-router-dom";
4 import { Toaster } from "sonner";
5
6 export default function DevTree(){
7   return(
8     <>

```

```

frontend > src > layouts > AppLayout.tsx > AppLayout
8   export default function AppLayout() {
10     const {data, isLoading, isError} = useQuery({
11       //pasarle algunas configuraciones
12       queryFn: getUser, //qué función va a hacer la consulta hacia nues
13       queryKey: ['user'], //reactquery identifica el query del getuser
14       retry: 1, //cuantas veces queremos que react intente la conexión
15       refetchOnWindowFocus: false //no queremos que se hagan mas consul
16     })
17
18     if(isLoading) return 'Cargando...'
19     if(isError){
20       return <Navigate to={'/auth/login'} />
21     }
22
23     //console.log(data)
24     //console.log(isLoading) //primero será false
25     //console.log(isError)
26     //console.log(error?.message)
27
28     //vía props le pasamos data
29     if(data) return <DevTree data=[data]>
30   }

```

Escribimos e importamos el type.

```

EXPLORER
  > THUNDER CLIENT: THUNDER CLIENT
    > backend
    < DEVTREE
      > frontend
        > node_modules
        > public
        < src
          > api
            TS DevTreeAPI.ts
          < components
            > DevTree.tsx 1
              TS DevTree.tsx
              TS ErrorMessage.tsx
              TS NavigationTabs.tsx
            < config
              TS axios.ts
            < layouts
              TS AppLayout.tsx
              TS AuthLayout.tsx

```

```

frontend > src > components > DevTree.tsx > DevTree
1 import React from 'react'
2 import NavigationTabs from "../components/NavigationTabs";
3 import { Link, Outlet } from "react-router-dom";
4 import { Toaster } from "sonner";
5 import type { User } from '../types';
6
7 type DevTreeProps = {
8   data: User
9 }
10
11 export default function DevTree({data}: DevTreeProps){
12   return(
13     <>
14       <header className="bg-slate-800 py-5">
15         <div className="mx-auto max-w-5xl flex flex-col md:flex-row items-center">
16           <div className="w-full p-5 lg:p-0 md:w-1/3">
17             
18           </div>
19           <div className="md:w-1/3 md:flex md:justify-end">

```

Quitamos importaciones sin utilizar.

Al hacer esta comprobación de if.. data, y haberlo colocado en su propio componente, podemos renderizar un valor.

```

10   export default function DevTree({data}: DevTreeProps){
29     <main className="mx-auto max-w-2xl p-10 mb:0" >
30       <NavigationTabs />
31
32       <div className="flex justify-end">
33         <Link
34           to={''}
35           target="_blank"
36           rel="noopener noreferrer"
37         >Visitar Mi Perfil: /{data.handle}</Link>
38
39       </div>
40
41       <div className="flex flex-col md:flex-row gap-10 mt-10">
42         <div className="flex-1">
43           <Outlet />
44         </div>
45         <div className="w-full md:w-96 bg-slate-800 px-5 py-10 space-y-6">
46
47       </div>
48

```

Trabajando en el Formulario de Edición de Perfil

Vamos a ver cómo llenar en automático el formulario de editar información. Si recuerdan, no tenemos una descripción en nuestro modelo:

Si recuerdas también, para la parte de formulario estamos usando ReactHookForm que con ReactQuery se llevan muy bien.

```

1 import {useForm} from 'react-hook-form'
2
3 export default function ProfileView() {
4
5   return (
6     <form
7       className="bg-white p-10 rounded-lg space-y-6"
8       onSubmit={() => {}}
9     >
10
11       <legend className="text-2xl font-bold text-slate-800">
12         <div className="grid grid-cols-1 gap-2">
13           <label
14             htmlFor="handle"
15             >Handle:</label>
16           <input
17             type="text"
18             className="border-none bg-slate-800 placeholder="handle o Nombre de usuario"

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

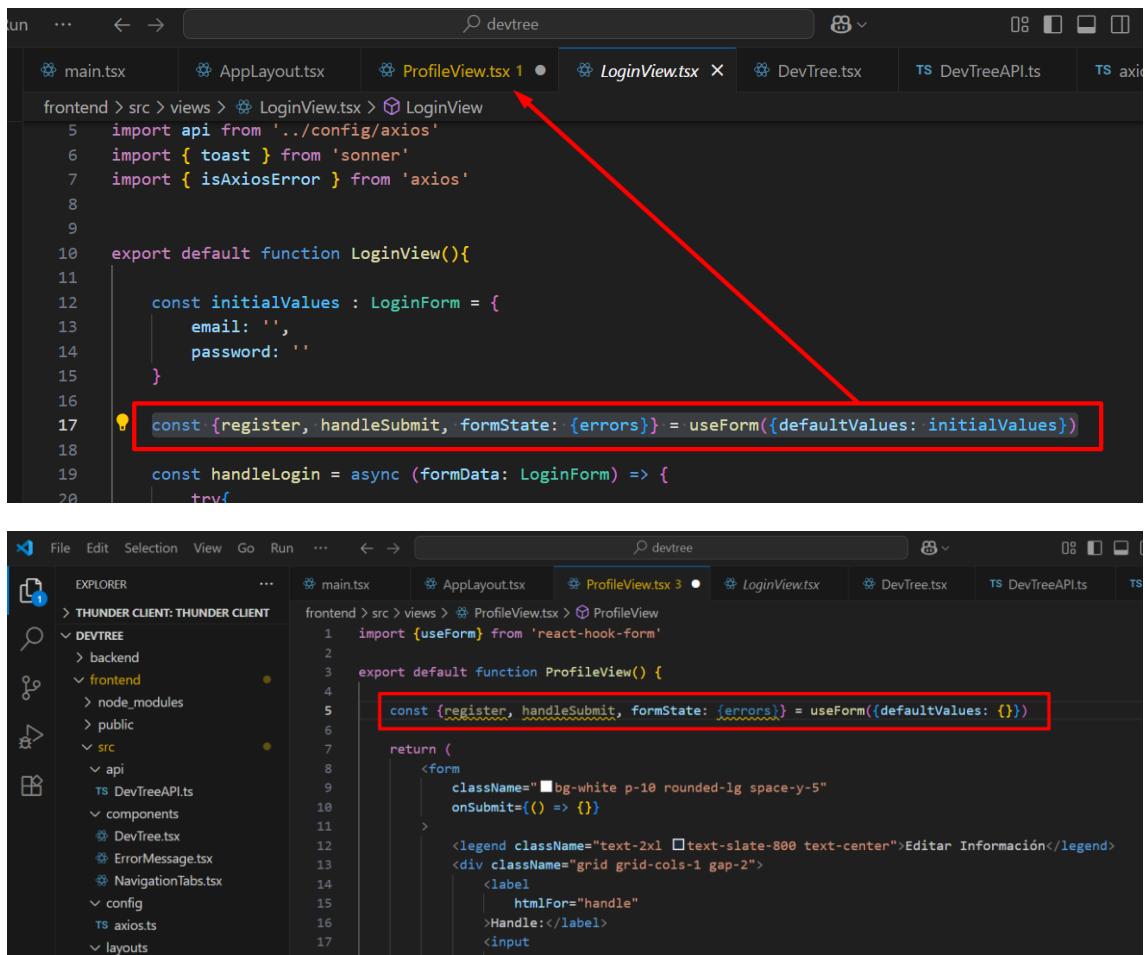
```

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

> devtree@1.0.0 dev
> nodemon src/index.ts

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*

```



```
frontend > src > views > LoginView.tsx
5 import api from '../config/axios'
6 import { toast } from 'sonner'
7 import { isAxiosError } from 'axios'
8
9
10 export default function LoginView(){
11
12     const initialValues : LoginForm = {
13         email: '',
14         password: ''
15     }
16
17     const {register, handleSubmit, formState: {errors}} = useForm({defaultValues: initialValues})
18
19     const handleLogin = async (formData: LoginForm) => {
20         try{

```

```
frontend > src > views > ProfileView.tsx
1 import {useForm} from 'react-hook-form'
2
3 export default function ProfileView() {
4
5     const {register, handleSubmit, formState: {errors}} = useForm({defaultValues: {}})
6
7     return (
8         <form
9             className="bg-white p-10 rounded-lg space-y-5"
10            onSubmit={() => {}}
11        >
12            <legend className="text-2xl text-slate-800 text-center">Editar Información</legend>
13            <div className="grid grid-cols-1 gap-2">
14                <label
15                    htmlFor="handle"
16                    >Handle:</label>
17                <input

```

Solo tenemos el handle, porque la descripción no existe en la API, pero vamos a trabajar con el handle, entonces vamos a colocar:

```

    import {useForm} from 'react-hook-form'
    import ErrorMessage from '../components/ErrorMessage'

    export default function ProfileView() {
      const {register, handleSubmit, formState: {errors}} = useForm({defaultValues: {handle: ''}})

      const handleUserProfileForm = () => {
        console.log('desde handleUserProfileForm')
      }

      return (
        <form
          className="bg-white p-10 rounded-lg space-y-5"
          onSubmit={handleSubmit(handleUserProfileForm)}
        >
          <legend>Editar Información</legend>
          <div className="grid grid-cols-1 gap-2">
            <label
              htmlFor="handle"
            >Handle:</label>
            <input
              type="text"
              className="border-none bg-slate-100 rounded-lg p-2"
              placeholder="handle o Nombre de Usuario"
              {...register('handle', {
                required: "El nombre de usuario es obligatorio."
              })}
            />
            {errors.handle && <ErrorMessage>{errors.handle.message}</ErrorMessage>}
          </div>
        </form>
      )
    }
  
```

The code editor highlights several parts of the code with red boxes and numbers:

- Line 1: A red box surrounds the `{...register('handle', { required: "El nombre de usuario es obligatorio." })}` line.
- Line 2: A red box surrounds the `const handleUserProfileForm = () => { console.log('desde handleUserProfileForm') }` line.
- Line 3: A red box surrounds the `{errors.handle && <ErrorMessage>{errors.handle.message}</ErrorMessage>}` line.
- Line 4: A red box surrounds the `onSubmit={handleSubmit(handleUserProfileForm)}` line.
- Line 5: A red box surrounds the `<form>` opening tag.

Guardamos cambios:

Links Mi Perfil

Visitar Mi Perfil: /rodrigocastilla

Editar Información

Handle:

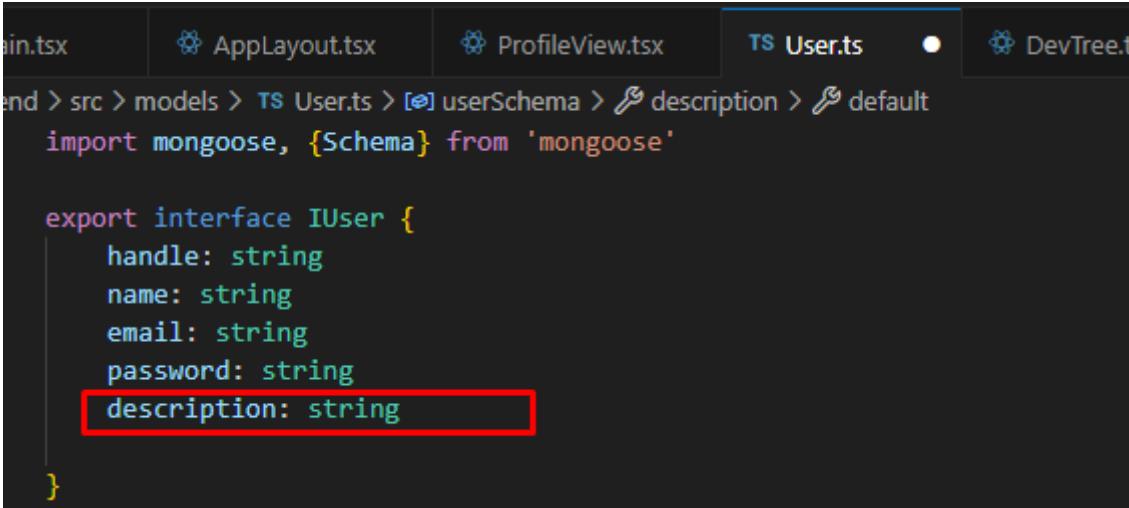
Tu Descripción

Imagen:

GUARDAR CAMBIOS

EL NOMBRE DE USUARIO ES OBLIGATORIO.

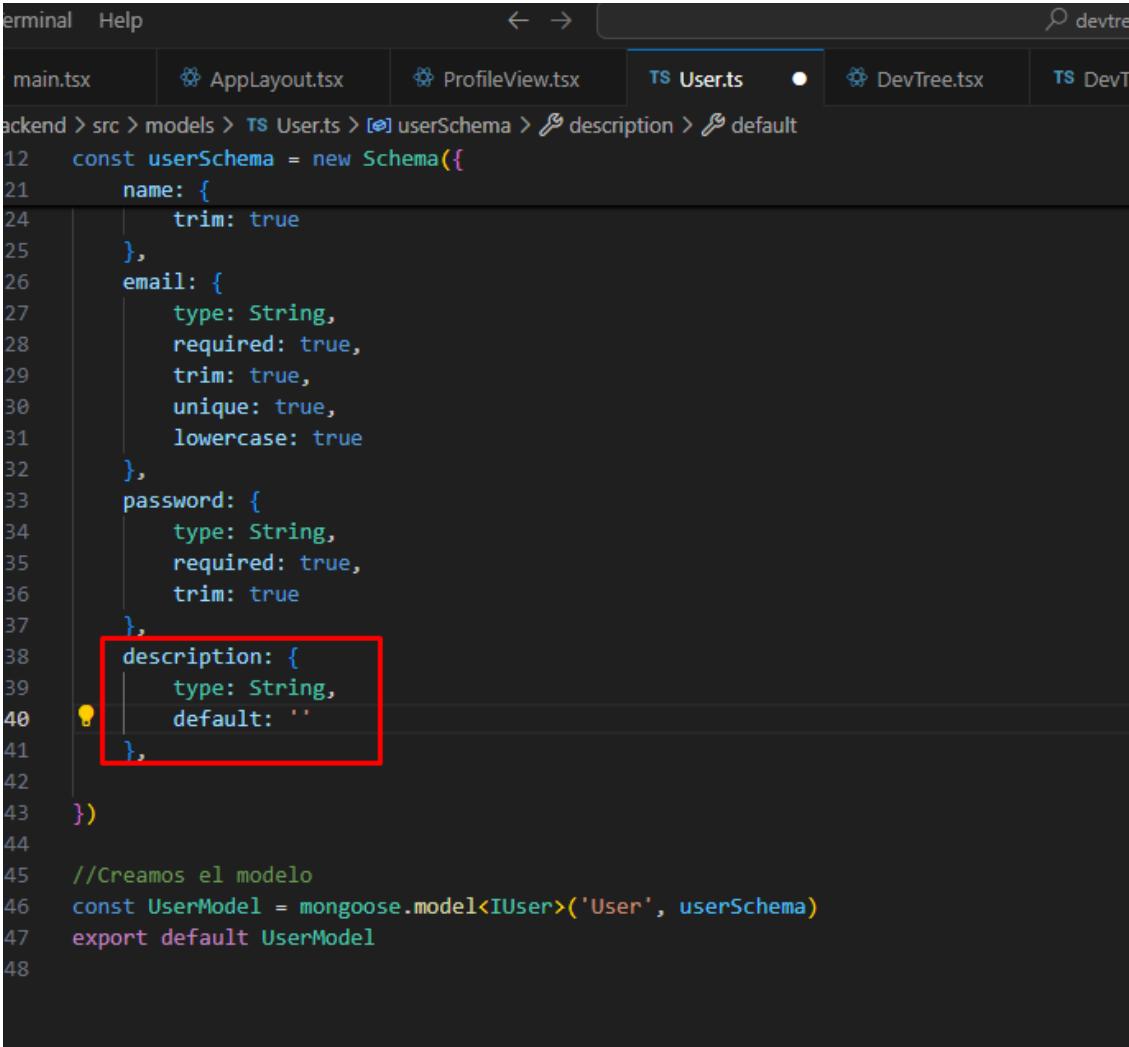
Vemos que tenemos un campo descripción, y ese campo no existe en nuestro modelo User.



```
main.tsx AppLayout.tsx ProfileView.tsx TS User.ts DevTree.tsx
end > src > models > TS User.ts > userSchema > description > default
import mongoose, {Schema} from 'mongoose'

export interface IUser {
  handle: string
  name: string
  email: string
  password: string
  description: string
}

}
```



```
terminal Help ← → devtree
main.tsx AppLayout.tsx ProfileView.tsx TS User.ts DevTree.tsx DevTree.tsx
ackend > src > models > TS User.ts > userSchema > description > default
12 const userSchema = new Schema({
21   name: {
24     trim: true
25   },
26   email: {
27     type: String,
28     required: true,
29     trim: true,
30     unique: true,
31     lowercase: true
32   },
33   password: {
34     type: String,
35     required: true,
36     trim: true
37   },
38   description: {
39     type: String,
40     default: ''
41   }
42 }
43 )
44
45 //Creamos el modelo
46 const UserModel = mongoose.model<IUser>('User', userSchema)
47 export default UserModel
48
```

Ahora, importante, estamos modificando el modelo, pero eso no quiere decir que estamos modificando lo que está en la BD. Dos opciones, lo dejamos como está, o

eliminamos el usuario de Compass. Y si recargamos, nos saca de la aplicación, volvemos a registrarnos. Y vemos que ya aparece “descripción” en la consola.

The screenshot shows a web application titled "DevTree" with a "Mi Perfil" (My Profile) section. On the left, there's a form for editing profile information, including fields for "Handle" (with placeholder "handle o Nombre de Usuario"), "Descripción" (with placeholder "Tu Descripción"), and "Imagen" (with a file selection input). A "GUARDAR CAMBIOS" (Save Changes) button is at the bottom. On the right, the DevTools console is open, displaying the JSON representation of a user object. The "description" field is highlighted with a red arrow. The JSON output is as follows:

```
{
  "id": "65cda858376d394252edc5d9",
  "handle": "rodrigocastilla",
  "name": "Rodrigo Castilla",
  "email": "rodrigocastilla@mail.com",
  "description": "", // Red arrow points here
  "handle": "rodrigocastilla"
}
```

Teníamos que eliminar el usuario, hacer cambio de modelo y crear otro para que traiga ese nuevo atributo.

The screenshot shows a code editor with several files listed in the tabs: main.tsx, AppLayout.tsx, ProfileView.tsx (the active file), User.ts, DevTree.tsx, DevTreeAPI.ts, and another partially visible file. The code in ProfileView.tsx is as follows:

```

import {useForm} from 'react-hook-form'
import ErrorMessage from '../components/ErrorMessage'

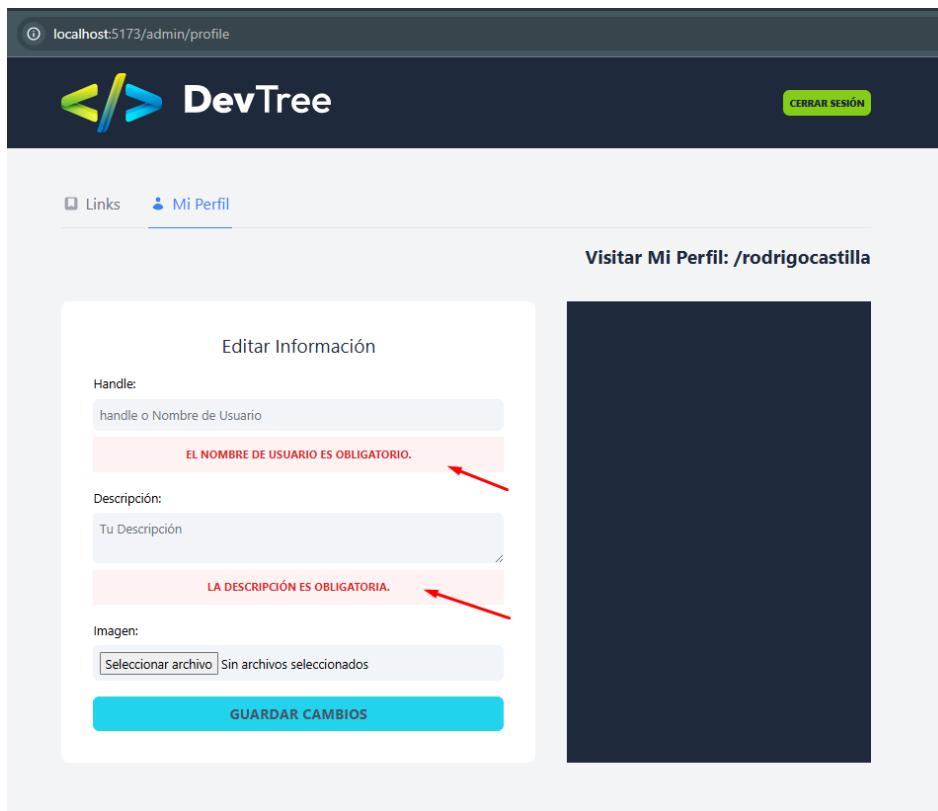
export default function ProfileView() {
  const {register, handleSubmit, formState: {errors}} = useForm({defaultValues: {
    handle: '',
    description: '' // Red arrow points here
  }})
  const handleUserProfileForm = () => {
    console.log('desde handleUserProfileForm')
  }
}

```

```

    main.tsx          AppLayouttsx      ProfileView.tsx •   TS Users.ts      TS DevTree.ts      TS DevTreeAPI.ts     TS axios.ts
frontend > src > views > ProfileView.tsx > ProfileView
  4  export default function ProfileView() {
    5    <form>
    6      <label htmlFor="handle">
    7        Handle:</label>
    8        <input
    9          type="text"
   10         className="border-none bg-slate-100 rounded-lg p-2"
   11         placeholder="Handle o Nombre de Usuario"
   12         {...register('handle', {
   13           required: "El nombre de usuario es obligatorio."
   14         })}
   15       />
   16       {errors.handle && <ErrorMessage>{errors.handle.message}</ErrorMessage>}
   17     </div>
   18
   19     <div className="grid grid-cols-1 gap-2">
   20       <label
   21         htmlFor="description">
   22           Descripción:</label>
   23       <textarea
   24         className="border-none bg-slate-100 rounded-lg p-2"
   25         placeholder="Tu Descripción"
   26         {...register('description', {
   27           required: "La descripción es obligatoria."
   28         })}
   29       />
   30       {errors.description && <ErrorMessage>{errors.description.message}</ErrorMessage>}
   31     </div>
   32
   33     <div className="grid grid-cols-1 gap-2">
   34       <label
   35         htmlFor="image">
   36           Imagen:</label>

```

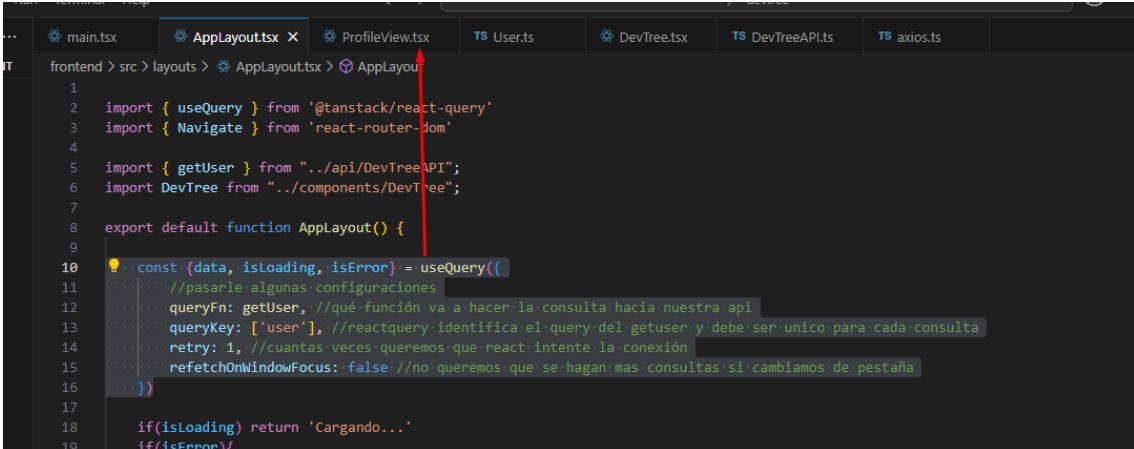


La imagen sí será opcional pero eso lo veremos después con Cloudinary.

Llenando el formulario y reutilizando consultas con React Query.

Vamos a ver como llenar el formulario con el handle, y descripción.

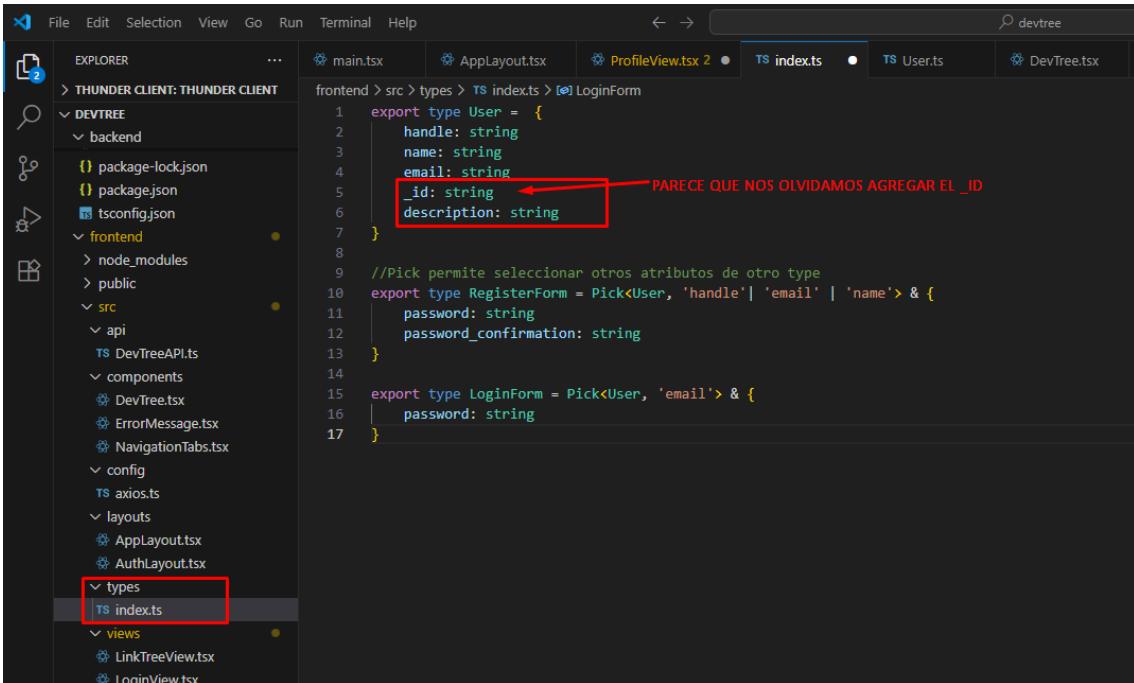
Copiar y llevarlo al ProfileView.



```

1 import { useQuery } from '@tanstack/react-query'
2 import { Navigate } from 'react-router-dom'
3
4 import { getUser } from "../api/DevTreeAPI"
5 import DevTree from "../components/DevTree"
6
7 export default function AppLayout() {
8     const { data, isLoading, isError } = useQuery({
9         //pasarle algunas configuraciones
10        queryFn: getUser, //qué función va a hacer la consulta hacia nuestra api
11        queryKey: ['user'], //reactquery identifica el query del getUser y debe ser unico para cada consulta
12        retry: 1, //cuantas veces queremos que react intente la conexión
13        refetchOnWindowFocus: false //no queremos que se hagan mas consultas si cambiamos de pestana
14    })
15
16    if(isLoading) return 'Cargando...'
17
18    if(isError)
19 }

```

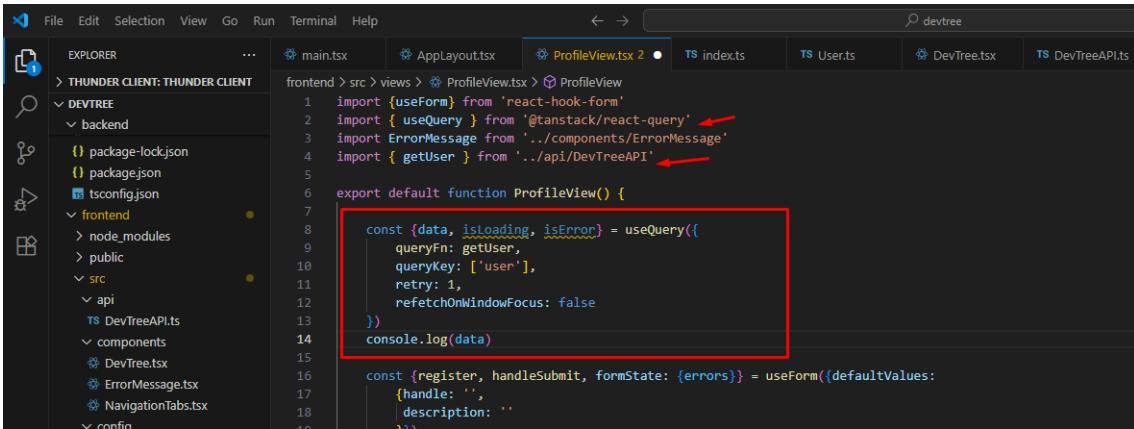


PARECE QUE NOS OLVIDAMOS AGREGAR EL _ID

```

1 export type User = {
2     handle: string
3     name: string
4     email: string
5     _id: string ← PARECE QUE NOS OLVIDAMOS AGREGAR EL _ID
6     description: string
7 }
8
9 //Pick permite seleccionar otros atributos de otro type
10 export type RegisterForm = Pick<User, 'handle' | 'email' | 'name' > & {
11     password: string
12     password_confirmation: string
13 }
14
15 export type LoginForm = Pick<User, 'email' > & {
16     password: string
17 }

```

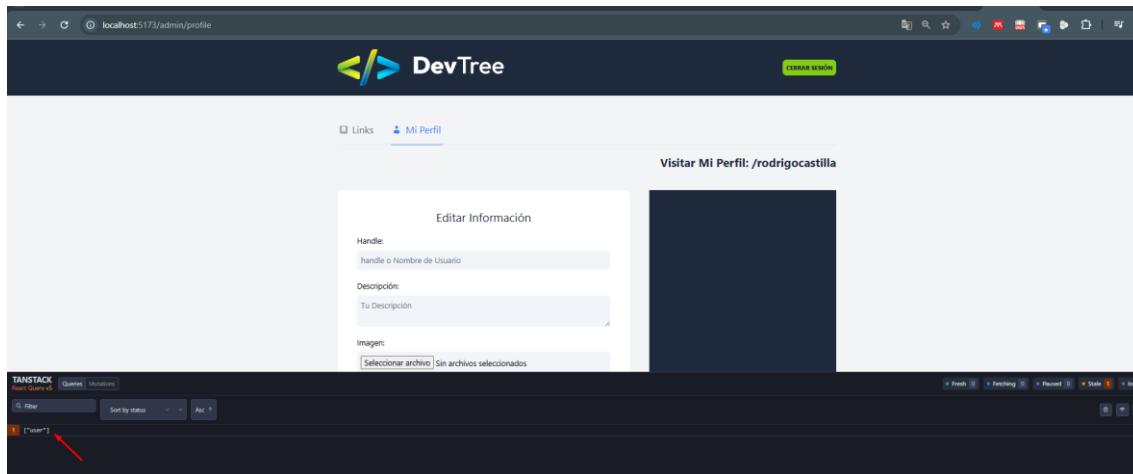


```

1 import { useForm } from 'react-hook-form'
2 import { useQuery } from '@tanstack/react-query'
3 import ErrorMessage from '../components/ErrorMessage'
4 import { getUser } from '../api/DevTreeAPI'
5
6 export default function ProfileView() {
7     const { data, isLoading, isError } = useQuery({
8         queryFn: getUser,
9         queryKey: ['user'],
10        retry: 1,
11        refetchOnWindowFocus: false
12    })
13    console.log(data)
14
15    const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
16        {handle: '',
17         description: ''}}
18 }

```

Ese código con UseQuery esta bien pero algo que vamos a notar en ReactQuery DevTools, es que tenemos este QueryPreview llamado user.

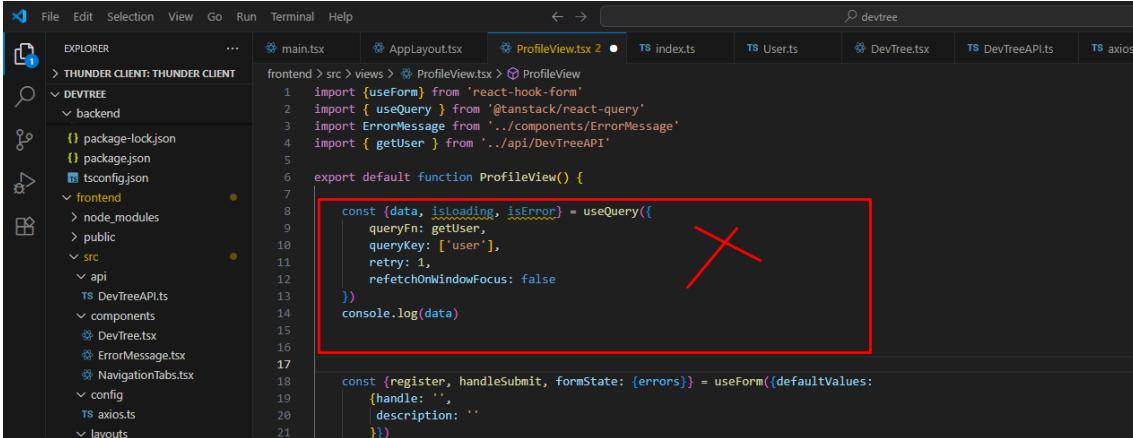


Entonces, si abrimos el App Layout, podemos ver que el query tiene user y en ProfileView lo estamos reutilizando.

En ReactQuery DevTools tenemos Fresh, Fetching, Pause, Stale, Inactive, el queryKey tiene color Naranja es decir es Stale, o sea es algo que ya está cacheado, es decir ya está en memoria y si recargamos tiene un flashazo de color azul y luego pasa a color naranja, tiene fetching es decir obtiene los datos y luego se coloca en caché.

Algo que hace bien reactquery, es que ya detectó el querykey no vuelve a hacer otro llamado al api si no toma los datos de lo que ya está cacheado, pero si tenemos acceso al panel, podemos evitar realizar la consulta y acceder a los datos cacheados que está en algún lugar en la memoria. Para eso podemos usar queryClient, que es lo que usamos para configurar reactquery de forma global, y dentro de tus componentes ellos te ofrecen un hook que te va a permitir acceder a los datos o a este objeto llamado queryclient, y hay un montón de información, una de esa es getQueryData y de esta forma podemos acceder a los datos cacheados.

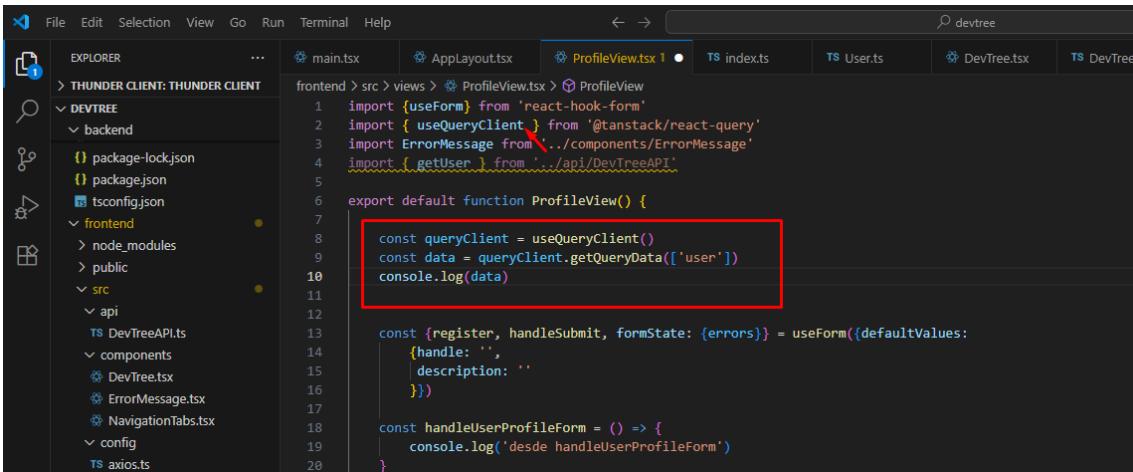
Eliminamos:



```

File Edit Selection View Go Run Terminal Help
EXPLORER main.tsx AppLayout.tsx ProfileView.tsx 2 TS index.ts TS User.ts DevTree.tsx DevTreeAPI.ts TS axios.ts
frontend > src > views > ProfileView.tsx > ProfileView
1 import {useForm} from 'react-hook-form'
2 import {useQuery} from '@tanstack/react-query'
3 import ErrorMessage from '../components/ErrorMessage'
4 import {getUser} from '../api/DevTreeAPI'
5
6 export default function ProfileView() {
7
8   const {data, isloading, isError} = useQuery({
9     queryFn: getUser,
10    queryKey: ['user'],
11    retry: 1,
12    refetchOnWindowFocus: false
13  })
14
15  console.log(data)
16
17  const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
18    {handle: '',
19     description: ''}})
19
20}
21

```

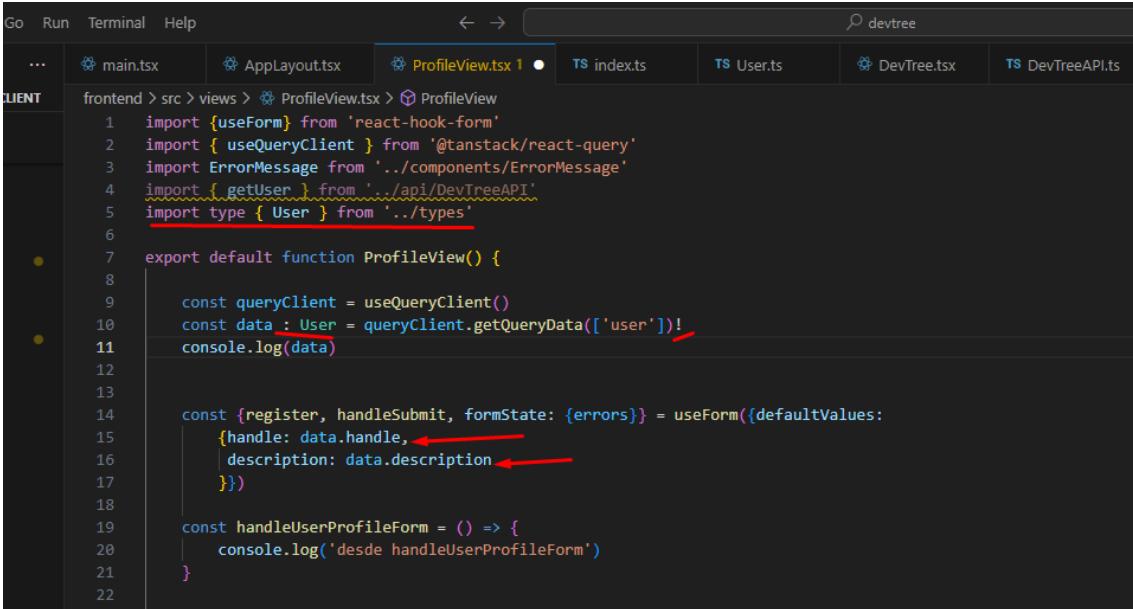


```

File Edit Selection View Go Run Terminal Help
EXPLORER main.tsx AppLayout.tsx ProfileView.tsx 1 TS index.ts TS User.ts DevTree.tsx DevTreeAPI.ts TS axios.ts
frontend > src > views > ProfileView.tsx > ProfileView
1 import {useForm} from 'react-hook-form'
2 import {useQueryClient} from '@tanstack/react-query'
3 import ErrorMessage from '../components/ErrorMessage'
4 import {getUser} from '../api/DevTreeAPI'
5
6 export default function ProfileView() {
7
8   const queryClient = useQueryClient()
9   const data = queryClient.getQueryData(['user'])
10  console.log(data)
11
12
13  const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
14    {handle: '',
15     description: ''}})
15
16  const handleUserProfileForm = () => {
17    console.log('desde handleUserProfileForm')
18}
19
20}
21

```

Esto hace lo mismo solo que más corto, ya que cuando tenemos dos UseQuery iguales y con el mismo querykey, no hace dos consultas, hace la primera y la segunda la obtiene de los datos cacheados.



```

Go Run Terminal Help
... main.tsx AppLayout.tsx ProfileView.tsx 1 TS index.ts TS User.ts DevTree.tsx DevTreeAPI.ts
CLIENT frontend > src > views > ProfileView.tsx > ProfileView
1 import {useForm} from 'react-hook-form'
2 import {useQueryClient} from '@tanstack/react-query'
3 import ErrorMessage from '../components/ErrorMessage'
4 import {getUser} from '../api/DevTreeAPI'
5 import type {User} from '../types'
6
7 export default function ProfileView() {
8
9   const queryClient = useQueryClient()
10  const data: User = queryClient.getQueryData(['user'])!
11  console.log(data)
12
13
14  const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
15    {handle: data.handle, // Red arrow here
16     description: data.description // Red arrow here}}
17  })
18
19  const handleUserProfileForm = () => {
20    console.log('desde handleUserProfileForm')
21}
22

```

Data ahora es de tipo User, le agregamos “:user”, pero igual puede ser undefined, y en caso que no existe nos otorgará un undefined, pero sabemos que va a existir,

porque si vamos a la URL verifica que estamos autenticados, pero TypeScript no sabe de eso, pero aquí podemos poner un signo de admiración, y esto lo que trata de decir es “te garantizo estoy seguro que va a existir ese user”.

```

File Edit Selection View Go Run Terminal Help
EXPLORER main.tsx AppLayout.tsx ProfileView.tsx 2 TS index.ts TS User.ts DevTree.tsx
THUNDER CLIENT: THUNDER CLIENT
DEVTREE
  backend
    package-lock.json
    package.json
    tsconfig.json
  frontend
    node_modules
    public
    src
      api
        DevTreeAPI.ts
      components
        DevTree.tsx
        ErrorMessage.tsx
        NavigationTabs.tsx
      config
        axios.ts
      layouts
        AppLayout.tsx
        AuthLayout.tsx
      types
        types.ts
frontend/src/views/ProfileView.tsx
1 import {useForm} from 'react-hook-form'
2 import {useQueryClient} from '@tanstack/react-query'
3 import ErrorMessage from '../components/ErrorMessage'
4 import {getUser} from '../api/DevTreeAPI'
5 import type {User} from '../types'
6
7 export default function ProfileView() {
8
9   const queryClient = useQueryClient()
10  const data: User = queryClient.getQueryData(['user'])!
11  console.log(data)
12
13  const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
14    {handle: data.handle,
15     description: data.description
16   })
17
18  const handleUserProfileForm = (formData: User) => {
19    console.log(formData)
20  }
21
22  return (
23    <form
24      className="bg-white p-10 rounded-lg space-y-5"

```

Ahora, tenemos ese formData, y si le agregamos el type de User, podemos hacerlo pero el formulario tiene más campos y el typeUser tiene solo 2.

Validación en el formulario Editar Perfil

Vamos a ver como agregar el type a formData.

```

File Edit Selection View Go Run Terminal Help
EXPLORER main.tsx AppLayout.tsx ProfileView.tsx 2 TS index.ts X TS User.ts DevTree.tsx
THUNDER CLIENT: THUNDER CLIENT
DEVTREE
  backend
    package-lock.json
    package.json
    tsconfig.json
  frontend
    node_modules
    public
    src
      api
        DevTreeAPI.ts
      components
        DevTree.tsx
        ErrorMessage.tsx
        NavigationTabs.tsx
      config
        axios.ts
      layouts
        AppLayout.tsx
        AuthLayout.tsx
      types
        index.ts
views
frontend/src/types/index.ts
1 export type User = {
2   handle: string
3   name: string
4   email: string
5   _id: string
6   description: string
7 }
8
9 //Pick permite seleccionar otros atributos de otro type
10 export type RegisterForm = Pick<User, 'handle' | 'email' | 'name'> & {
11   password: string
12   password_confirmation: string
13 }
14
15 export type LoginForm = Pick<User, 'email'> & {
16   password: string
17 }
18
19 export type ProfileForm = Pick<User, 'handle' | 'description'>

```

```

1 import {useForm} from 'react-hook-form'
2 import {useQueryClient} from '@tanstack/react-query'
3 import ErrorMessage from '../components/ErrorMessage'
4 import {getIsen} from '../api/DevTreeAPI'
5 import type {ProfileForm, User} from '../types'
6
7 export default function ProfileView() {
8
9     const queryClient = useQueryClient()
10    const data: User = queryClient.getQueryData(['user'])!
11    console.log(data)
12
13    const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
14        {handle: data.handle,
15         description: data.description
16     }})
17
18    const handleUserProfileForm = (formData: ProfileForm) => {
19        console.log(formData)
20    }
21}
22

```

Ahora, si pasamos el mouse por encima de "register" vemos que infiere que tiene dos datos: handle y description (esto por los valores previos de los datos cacheados).

```

const queryClient = useQueryClient()
const data: User = queryClient.getQueryData(['user'])!
console.log(data)
const {register, handleSubmit, formState: {errors}} = useForm({defaultValues:
    {handle: data.handle,
     description: data.description
    }})
const handleUserProfileForm = (formData: ProfileForm) => {
    console.log(formData)
}

```

Pero podemos ponerle via generic que tenga un type de ProfileForm.

	Type	Description
_id	ObjectId	ObjectID
handle	String	String
name	String	Rodrigo Castilla
email	String	rodrigocastilla@gmail.com
password	String	52b5108wruLN.zv1G2zl9aXsujfeRVL/cQn8Dt.mohP6ipbITE5yaoEMe
description	String	Esta es la descripción del usuario..:/
__v	Int32	0

Document modified. CANCEL UPDATE

Si queremos agregar o cambiarle la información de la descripción, tenemos que crear un nuevo endpoint en nuestro backend.

Del profileview.tsx hay que borrar `import { getUser } from './api/DevTreeAPI'`

Creando el endpoint en el backend para actualizar el perfil

```

File Edit Selection View Go Run Terminal Help devtree
EXPLORER
> THUNDER CLIENT: THUNDER CLIENT
  < DEVTREE
    < backend
      < dist
      < node_modules
      < src
        < config
        < handlers
          < TS index.ts
          > middleware
          < models
          < User.ts
          < utils
          < auth.ts
          < jwt.ts
          < index.html
          < index.ts
          < router.ts
          < server.ts
        < .env
        < package-lock.json
        < package.json
        tsconfig.json
      > frontend
TS index.ts •
backend > src > handlers > TS index.ts > updateProfile
46  export const login = async(req: Request, res: Response) => {
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
  //Comprobar el password
  const isPasswordCorrect = await checkPassword(password, user.password)
  if(!isPasswordCorrect){
    const error = new Error('Password incorrecto')
    return res.status(401).json({error: error.message})
    //401: porque no está autorizado a acceder a este recurso
  }

  //Hasta aquí tenemos un usuario que ingresó correctamente, entonces:
  const token = generateJWT({id: user._id})
  res.send(token)

}

export const getUser = async(req: Request, res: Response) => {
  res.json(req.user)
}

export const updateProfile = async(req: Request, res: Response) => {
  try{
    console.log(req.body)
  }catch(e){
    const error = new Error("Hubo un error")
    return res.status(500).json({error: error.message})
  }
}

```

```

File Edit Selection View Go Run Terminal Help
EXPLORER TS index.ts TS routers.ts
> THUNDER CLIENT: THUNDER CLIENT
> DEVTREE
  > backend
    > dist
    > node_modules
  > src
    > config
    > handlers
    > middleware
    > models
    > utils
  > index.html
  > TS index.ts
  > TS routers.ts
  > TS server.ts
  > .env
  > package-lock.json
  > package.json
  > tsconfig.json
  > frontend
  > TS routers.ts
  > TS server.ts
  > .env
  > package-lock.json
  > package.json
  > tsconfig.json
  > frontend

TS routers.ts
backend > src > TS routers.ts > (e) default
23   .withMessage('Email no valido'),
24
25   body('password')
26   .notEmpty()
27   .withMessage("El password es obligatorio"),
28
29   login
30
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 de la petición
37 //patch: aplica modificaciones parciales a un recurso (sería mejor porque sólo vamos a cambiar el handle y su descripción)
38
39 //para modificar un usuario necesitamos que esté autenticado por eso el authenticate
40 router.patch('/user', authenticate, updateProfile)
41
42 export default router
43
44

```

En Postman, creamos una nueva petición de tipo Patch. Activamos Bearer Token.

The screenshot shows the Postman interface with the following details:

- Request Type:** PATCH
- URL:** http://localhost:4000/user
- Authorization:** Bearer Token (selected)
- Body (Raw JSON):**

```

{
  "email": "rodrigocastilla@gmail.com",
  "password": "12345678"
}

```

Hay que autenticar el usuario, para refrescar el token, lo copiamos.

The screenshot shows the Postman interface with the following details:

- Request Type:** POST
- URL:** http://localhost:4000/auth/login
- Body (Raw JSON):**

```

{
  "email": "rodrigocastilla@gmail.com",
  "password": "12345678"
}

```

- Response Status:** 200 OK
- Response Body (Raw JSON):**

```

eyJhbGciOiJIUzI1NiIsInR6cC16IkxVCj9.eyJpZCI6IjY4NWRhODUwMzc2ZDM5NDI1MmFkYzViZCisImIhdCI6MTc1MDk3NjM3NCwiZXhwIjoxNzY2NTI4Mzc0fQ.l6xyNmhhyzP_GE-01ScxSC14hKyYo8mGsumqlsNdn8

```

Copiamos el token:

DevTree / Obtener Usuario Autenticado

GET http://localhost:4000/user

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Body (raw, JSON, XML, etc.)

```
{
  "handle": "rodrigocastilla@gmail.com",
  "description": "Esta es la descripción del usuario..."
}
```

Response (JSON)

```
{
  "_id": "606da880376d394252ad5bd",
  "handle": "rodrigocastilla",
  "name": "Rodrigo Castilla",
  "email": "rodrigocastilla@gmail.com",
  "description": "Esta es la descripción del usuario...",
  "__v": 0
}
```

```

TS index.ts TS router.ts
backend > src > TS router.ts > default
27     .withMessage('El password es obligatorio'),
28
29     login
30   )
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 de la petición
37   //patch: aplica modificaciones parciales a un recurso (sería mejor porque solo vamos a cambiar el handle y su descripción)
38
39   //para modificar un usuario necesitamos que esté autenticado por eso el authenticate,
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   export default router
50
51

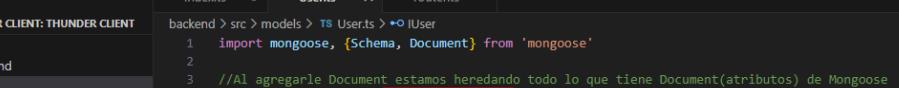
```

Ahora veamos cuál es la lógica para cambiar la información del perfil del usuario:

Permitir al usuario cambiar su handle

```
File Edit Selection View Go Run Terminal Help ⌘ devtree
EXPLORER ... TS index.ts TS router.ts
backend > src > handlers > TS indexts > [D] updateProfile
46 export const login = async(req: Request, res: Response) => {
47   const token = generateJWT([id: user._id])
48   res.send(token)
49 }
50
51 export const getUser = async(req: Request, res: Response) => {
52   res.json(req.user)
53 }
54
55 export const updateProfile = async(req: Request, res: Response) => {
56   try{
57     //console.log(req.body)
58     const {description} = req.body
59
60     //copiamos de arriba: esto porque el usuario no puede utilizar un usuario que est a utilizando otra persona
61     const handle = slug(req.body.handle, '')
62     const handleExists = await User.findOne({handle})
63     if(handleExists){
64       const error = new Error('Nombre de usuario no disponible')
65       return res.status(409).json({error: error.message})
66     }
67
68     //Actualizar el usuario
69     req.user.description = description
70     req.user.handle = handle
71
72   }catch(e){
73     const error = new Error("Hubo un error")
74     res.status(500).json({error: error.message})
75   }
76 }
```

Ahora, si recordamos, el middleware de auth, le decimos req.user tiene que ser la Interfaz de Usuario, y ahora queremos usar .save y no sabe que existe, ya que tiene otros atributos. Debo tener acceso a todos los métodos de Mongoose, pero no lo tenemos porque la interfaz solo tiene atributos:



```
import mongoose, {Schema, Document} from 'mongoose'
export interface IUser extends Document {
  handle: string
  name: string
  email: string
  password: string
  description: string
}
const userSchema = new Schema({
  //Atributos
  handle: {
    type: String,
    unique: true
  }
})
```

Vamos a probarlo:

The screenshot shows the Postman interface with the following details:

- Header:** PATCH / http://localhost:4000/user
- Body:** JSON (Raw) content:

```
1 [  
2   "handle": "rodrigocastilla@gmail.com",  
3   "description": "Descripción actualizada.",  
4 ]
```
- Response:** 200 OK | 406 ms - 274 B | Save Response

A red box highlights the "Send" button, and another red arrow points to it from the right. A third red box highlights the "description" field in the JSON body.

The screenshot shows the Postman interface with the following details:

- Header:** PATCH, http://localhost:4000/user
- Body (JSON):**

```

1  {
2     "handle": "rodrigocastilla@gmail.com",
3     "description": "Descripción actualizada."
4 }
```

- Response Body:** Perfil actualizado correctamente (highlighted by a red arrow).

The screenshot shows the DevTree application interface with the following details:

- Header:** CERRAR SESIÓN
- Links:** Links, Mi Perfil (highlighted by a blue underline).
- Text:** Visitar Mi Perfil: /rodrigocastilla@gmail.com
- Form:** Editar Información (Modal)
 - Handle:** rodrigocastilla@gmail.com
 - Descripción:** Descripción actualizada.
 - Imagen:** Seleccionar archivo Sin archivos seleccionados
 - Button:** GUARDAR CAMBIOS
- Image:** A large dark placeholder image where the profile picture would normally appear.

Ahora, puede salirnos un error de nombre de usuario no disponible. La API no sabe que somos nosotros mismos, entonces:

```

export const updateProfile = async(req: Request, res: Response) => {
  try{
    //console.log(req.body)
    const {description} = req.body

    //copiamos de arriba: esto porque el usuario no puede utilizar un usuario que está utilizando otra persona
    const handle = slug(req.body.handle, '')
    const handleExists = await User.findOne({handle})
    if(handleExists && handleExists.email !== req.user.email){
      const error = new Error('Nombre de usuario no disponible')
      return res.status(400).json({error: error.message})
    }

    //Actualizar el usuario
    req.user.description = description
    req.user.handle = handle
  }
}

```

elonMusk - elon@x.com

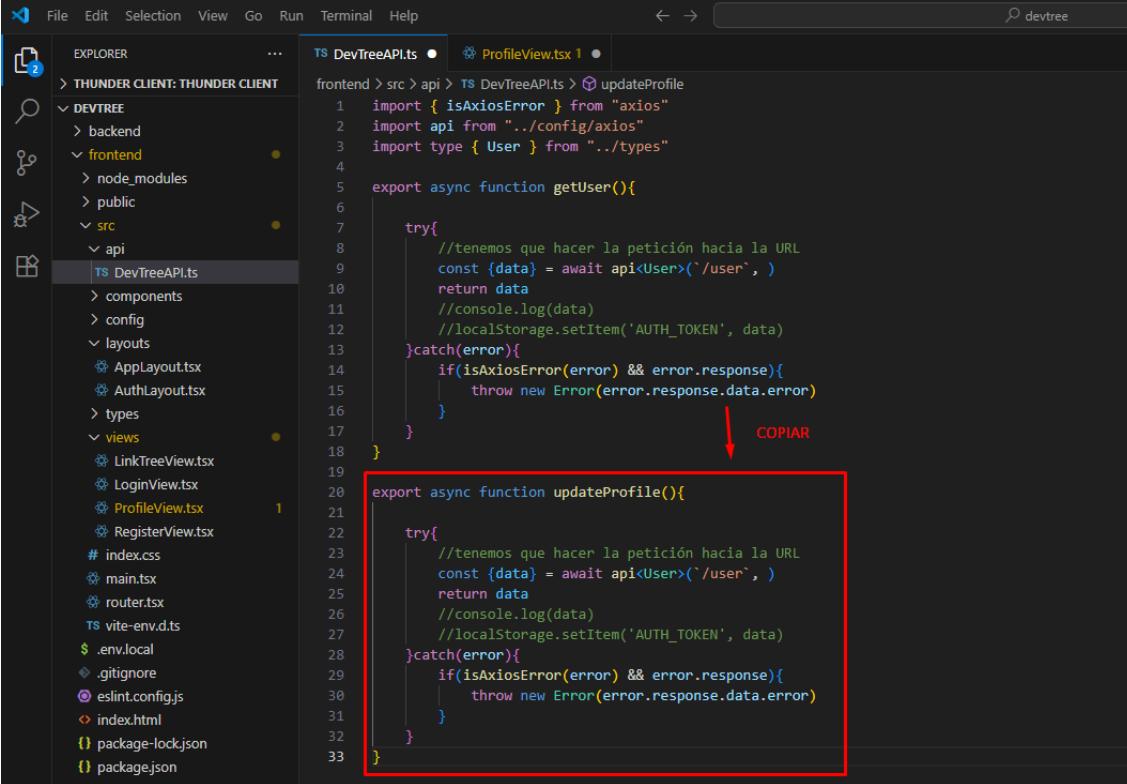
el_lobo@gmail.com que quiere ser Elon

el email elon@x.com tiene el handle de ElonMusk, entonces el_lobo no puede usar ese handle.

Hacemos la prueba.

Escribiendo la función en React que actualizará el usuario

Vamos al front. updateProfile



```

TS DevTreeAPI.ts ● ProfileView.tsx 1 ●
frontend > src > api > TS DevTreeAPI.ts > updateProfile
  1 import { isAxiosError } from "axios"
  2 import api from "../config/axios"
  3 import type { User } from "../types"
  4
  5 export async function getUser(){
  6
  7   try{
  8     //tenemos que hacer la petición hacia la URL
  9     const {data} = await api<User>('/user', )
 10    return data
 11    //console.log(data)
 12    //localStorage.setItem('AUTH_TOKEN', data)
 13  }catch(error){
 14    if(isAxiosError(error) && error.response){
 15      throw new Error(error.response.data.error)
 16    }
 17  }
 18}
 19
 20 export async function updateProfile(){
 21
 22   try{
 23     //tenemos que hacer la petición hacia la URL
 24     const {data} = await api<User>('/user', )
 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 }

```

Usamos useQuery para las peticiones de tipo GET, y las de POST utilizan useMutation.

The screenshot shows the VS Code interface with the DevTree API project open. The Explorer sidebar on the left displays the project structure, including the `THUNDER CLIENT`, `DEVTREE`, `frontend`, `src`, and `views` folders. The `ProfileView.tsx` file is selected in the Explorer, and its content is shown in the main code editor area.

```
TS DevTreeAPI.ts • ProfileView.tsx |
```

```
frontend > src > views > ProfileView.tsx > handleUserProfileForm
1 import {useForm} from 'react-hook-form'
2 import {useQueryClient, useMutation} from '@tanstack/react-query'
3 import ErrorMessage from './components/ErrorMessage'
4 import type {ProfileForm, User} from '../types'
5 import {updateProfile} from '../api/DevtreeAPI'
6
7 export default function ProfileView() {
8
9     const queryClient = useQueryClient()
10    const data : User = queryClient.getQueryData(['user'])!
11    console.log(data)
12
13
14    const {register, handleSubmit, formState: {errors}} = useForm<ProfileForm>({defaultValues:
15        {handle: data.handle,
16         description: data.description
17     }})
18
19    const updateProfileMutation = useMutation({
20        mutationFn: updateProfile,
21        onError: ()=>{
22            console.log('Hubo un error')
23        },
24        onSuccess: ()=>{
25            console.log('Todo bien')
26        }
27    })
28
29    const handleUserProfileForm = (FormData: ProfileForm) =>
```

¿En qué momento vamos a mutar los datos? Cuando pasemos la validación de lado del cliente.

```
const updateProfileMutation = useMutation({
    mutationFn: updateProfile,
    onError: ()=>{
        console.log('Hubo un error')
    },
    onSuccess: ()=>{
        console.log('Todo bien')
    }
})

const handleUserProfileForm = (formData: ProfileForm) => {
    //console.log(formData)
    updateProfileMutation.mutate(formData)
}

return (

```

Nos sale un error, eso pasa porque copiamos de getUser que no toma ningún parámetro, pero updateprofile si toma parámetros:

```
... TS DevTreeAPI.ts • ProfileView.tsx • ENT frontend > src > api > DevTreeAPI.ts > updateProfile
1 import { isAxiosError } from "axios"
2 import api from "../config/axios"
3 import type { ProfileForm, User } from "../types"
4
5 export async function getUser(){
6
7     try{
8         //tenemos que hacer la petición hacia la URL
9         const {data} = await api<User>(`/user`, )
10        return data
11        //console.log(data)
12        //localStorage.setItem('AUTH_TOKEN', data)
13    }catch(error){
14        if(isAxiosError(error) && error.response){
15            throw new Error(error.response.data.error)
16        }
17    }
18 }
19
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 }
```

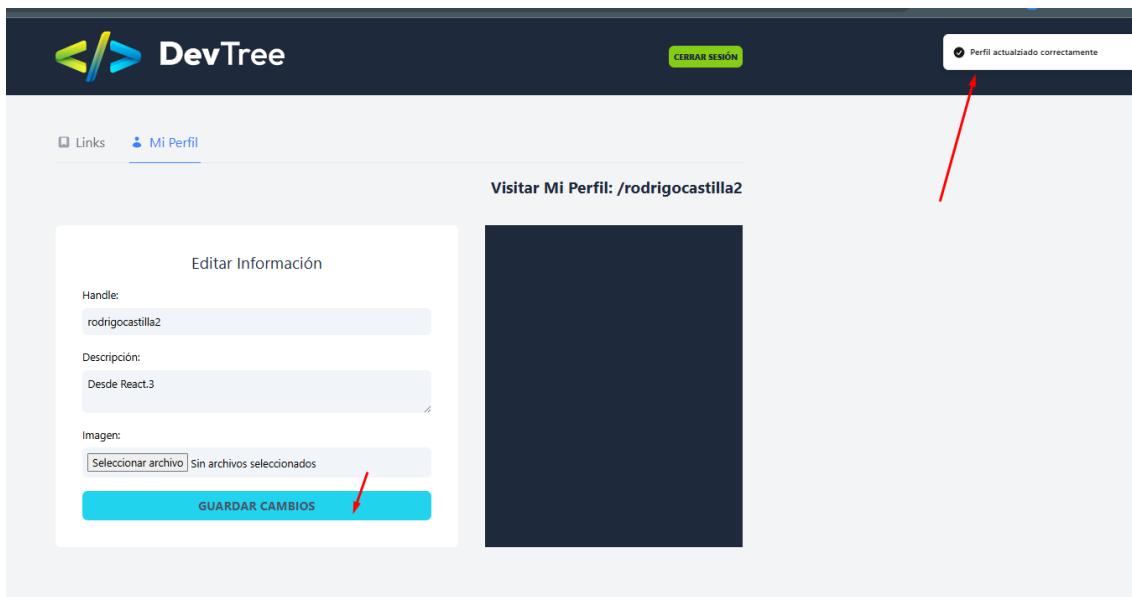
The screenshot shows the DevTree application interface. On the left, there's a sidebar with 'Links' and 'Mi Perfil'. The main area has a title 'Visitar Mi Perfil: /rodrigocastilla@gmail.com' and a form titled 'Editar Información'. The form contains fields for 'Handle' (rodrigocastilla@gmail.com), 'Descripción' (Desde React. with a red arrow pointing to it), and an 'Imagen' section with a file input field ('Seleccionar archivo Sin archivos seleccionados'). A large blue button at the bottom says 'GUARDAR CAMBIOS' with a red arrow pointing to it. To the right, a browser window shows the DevTools Network tab with several network requests listed, including one for 'rodrigocastilla@gmail.com' with a red arrow pointing to it.

Actualizando la información y mostrando un Toast

```

File Edit Selection View Go Run Terminal Help
EXPLORER > THUNDER CLIENT: THUNDER CLIENT
DEVTREE > backend > frontend > node_modules > public > src > api > DevTreeAPI.ts > components > config > layouts > AppLayout.tsx > AuthLayout.tsx > types > views > LinkTreeView.tsx > LoginView.tsx > ProfileView.tsx > RegisterView.tsx # index.css & main.tsx & router.tsx TS vite-env.d.ts $ .env.local & .gitignore
TS DevTreeAPI.ts ProfileView.tsx
frontend > src > views > ProfileView > updateProfileMutation > onSuccess
1 import { useForm } from 'react-hook-form'
2 import { useQueryClient, useMutation } from '@tanstack/react-query'
3 import { toast } from 'sonner'
4 import ErrorMessage from '../components/ErrorMessage'
5 import type { ProfileForm, User } from '../types'
6 import { updateProfile } from '../api/DevTreeAPI'
7
8 export default function ProfileView() {
9
10    const queryClient = useQueryClient()
11    const data : User = queryClient.getQueryData(['user'])!
12    console.log(data)
13
14    const {register, handleSubmit, formState: {errors}} = useForm<ProfileForm>({defaultValues:
15        {handle: data.handle,
16         description: data.description
17     }})
18
19    const updateProfileMutation = useMutation({
20        mutationFn: updateProfile,
21        onError: ()=>{
22            console.log('Hubo un error')
23        },
24        onSuccess: (data)=>{
25            toast.success(data)
26        }
27    })
28
29 }

```



Ese mensaje viene desde el servidor.

```

const updateProfileMutation = useMutation({
    mutationFn: updateProfile,
    onError: (error)=>{
        console.log(error.message)
    },
    onSuccess: (data)=>{
        toast.success(data)
    }
})

```

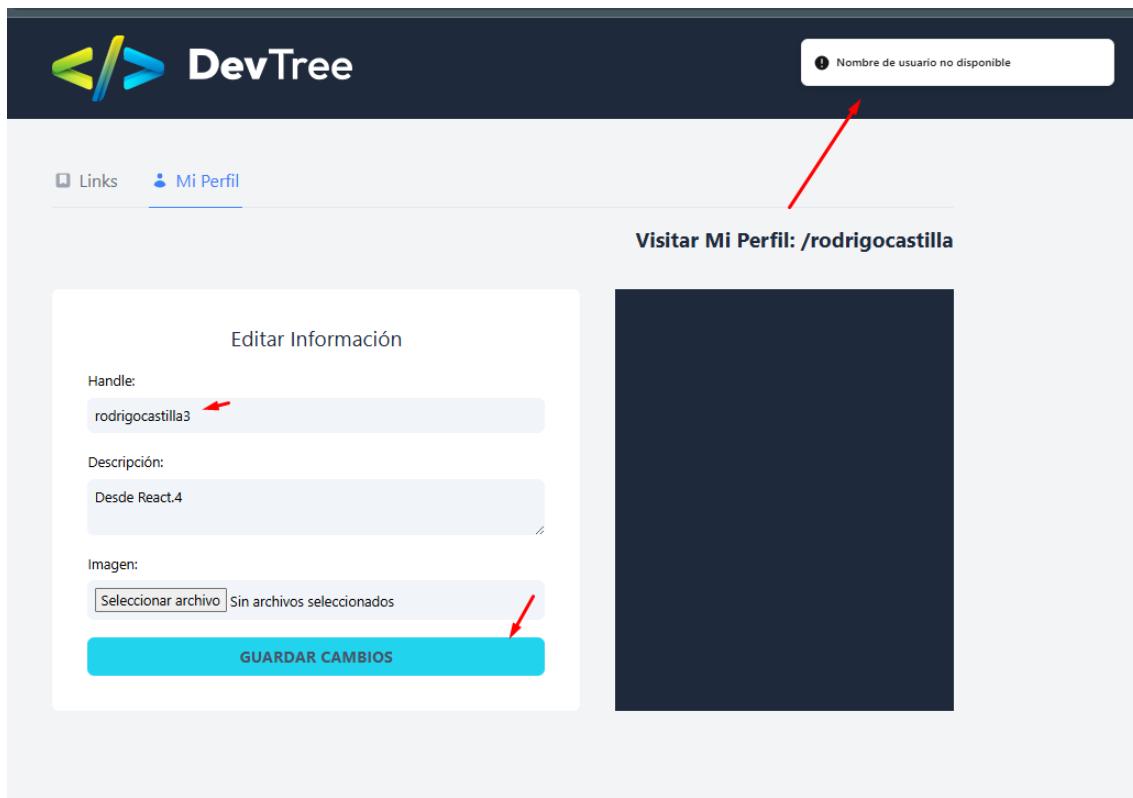
Si guardo cambios con otro usuario que no es rodrigocastilla, sale el error Nombre de usuario no disponible. En Compass debo tener dos usuarios creados.

The screenshot shows a web application interface titled "DevTree". On the left, there's a sidebar with "Links" and "Mi Perfil". The main area has a title "Visitar Mi Perfil: /rodrigocastilla". A modal window titled "Editar Información" is open, showing fields for "Handle" (set to "rodrigocastilla3"), "Descripción" (set to "Desde React.4"), and "Imagen" (with a placeholder "Seleccionar archivo Sin archivos seleccionados"). A red box highlights the "Handle" field, and a red arrow points to the "GUARDAR CAMBIOS" button. To the right, the browser's developer tools "Console" tab is open, displaying multiple entries for a user with handle "rodrigocastilla@gmail.com". One entry shows a warning: "PATCH http://localhost:4000/user 409 (Conflict) Nombre de usuario no disponible".

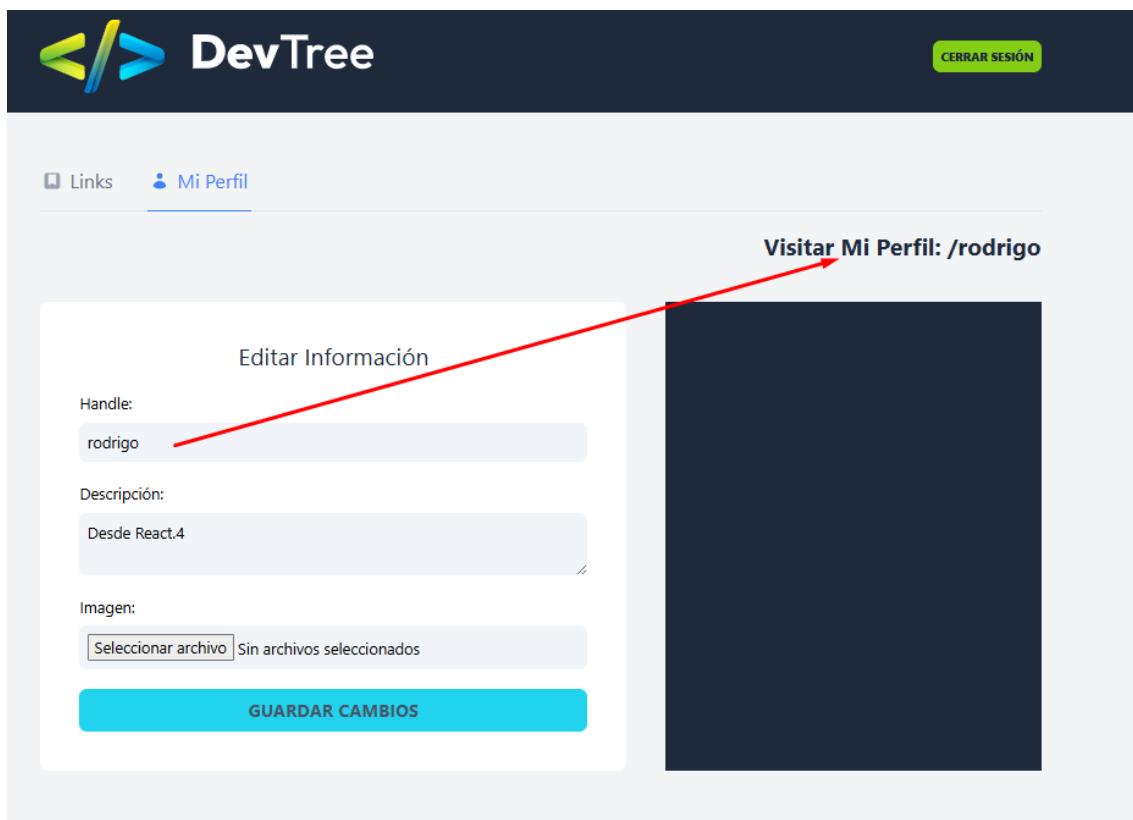
Ahora:

```
const updateProfileMutation = useMutation({
  mutationFn: updateProfile,
  onError:(error)=>{
    toast.error(error.message)
  },
  onSuccess:(data)=>{
    toast.success(data)
  }
})
```

```
const handleUserProfileForm = (formData: ProfileForm) => {
```



Ahora, si cambio el handle y guardo cambios, arriba en visitar mi perfil no se actualiza de forma automática, hay que recargar la página.

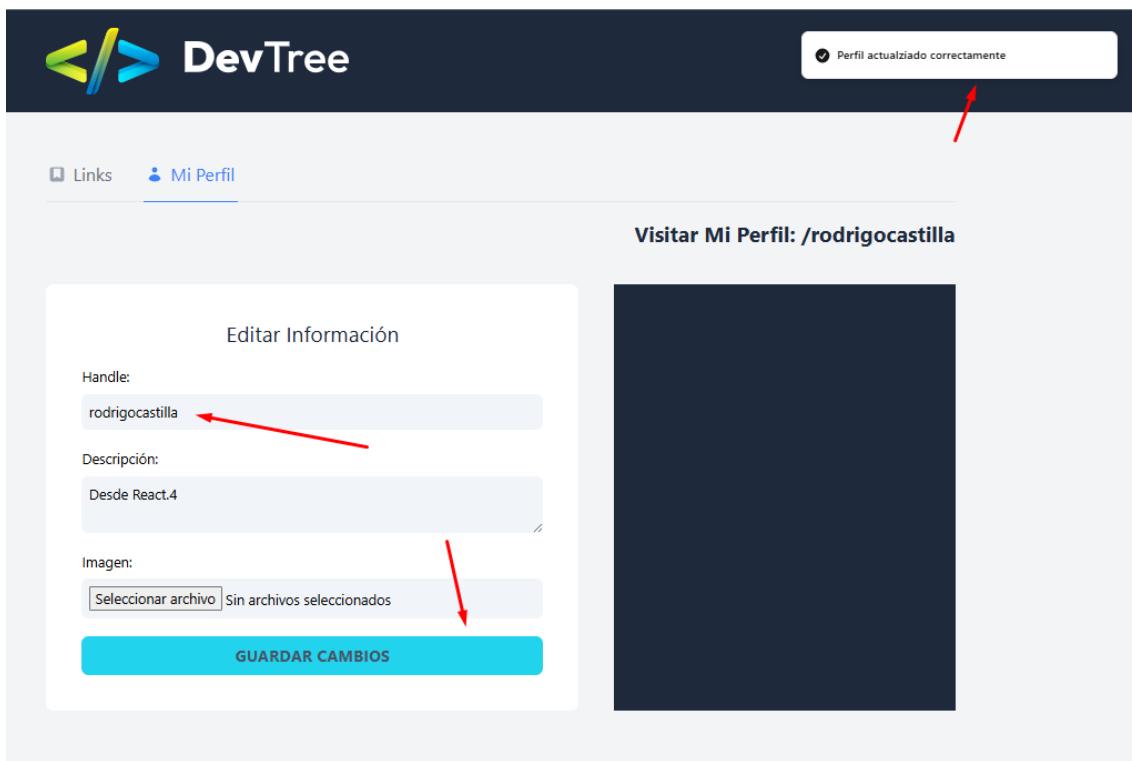


Veamos.

Invalidar los datos cacheados y hacer un re-fetch con React-Query

Usaremos queryInvalidation.

```
const updateProfileMutation = useMutation({
  mutationFn: updateProfile,
  onError:(error)=>{
    toast.error(error.message)
  },
  onSuccess:(data)=>{
    toast.success(data)
    //si se actualiza correctamente invalidamos el query y hace otra
    //consulta hacia la api y trae la info actualizada y la interfaz también
    queryClient.invalidateQueries({queryKey: ['user']})
  }
})
```



Lo hace automático porque elimina los datos cacheados y vuelve a ejecutar la función que se encarga de obtener el perfil del usuario.