Actividad 3.1: Bcrypt

Álvaro Del Valle Fernández

Índice

1. Introducción

En esta práctica debo diferenciar hash, cifrado y encoding. Instalar y usar bcrypt para las contraseñas y las api keys y entender como funciona el login con JWT (JSON Web Token).

2. Diferencias hash, cifrado y encoding

Hash convierte las contraseñas en una cadena alfanumerica usando algoritmos, es irreversible, de esto se encarga Bcrypt. Cifrado es la accion que transforma los datos en algo ilegible para protegerlos, pero es reversible. Encoding es la accion de convertir datos para que sean legibles por los sistemas, es reversible y no seguro. Por ejemplo convertir una linea de texto a binario.

3. Aplicar Bcrypt

Bcrypt es un algoritmo de hashing de contraseñas, diseñado para proteger credenciales. Bcrypt ya esta implementado, por lo que los siguientes pasos se centraran en probar su correcto funcionamiento.

4. Configuración

Tras clonar el repositorio uso

```
npm install
```

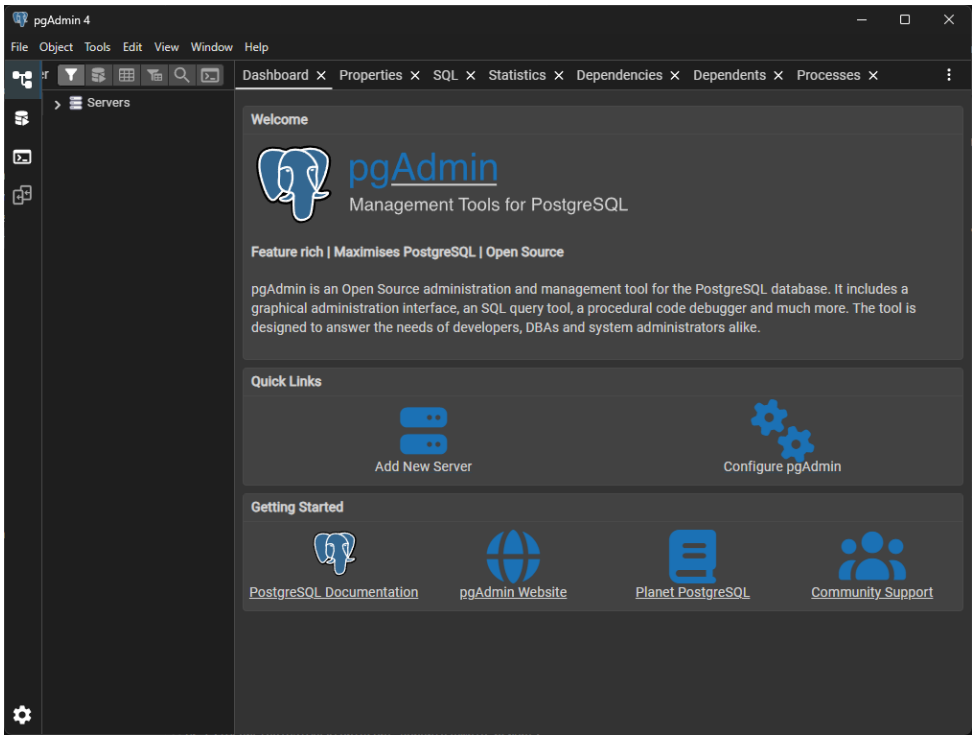
Creo las variables de entorno:

```
PS C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt> copy .env.
    example .env
copy : Cannot find path 'C:\Users\delva\Desktop\Bcypt Rober\
    ejemplo-bcrypt\.env.example' because it does not exist.
At line:1 char:1
+ copy .env.example .env
+ ~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (C:\Users\delva
        \...pt\.env.example:String) [Copy-Item],
        ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.
        Commands.CopyItemCommand
```

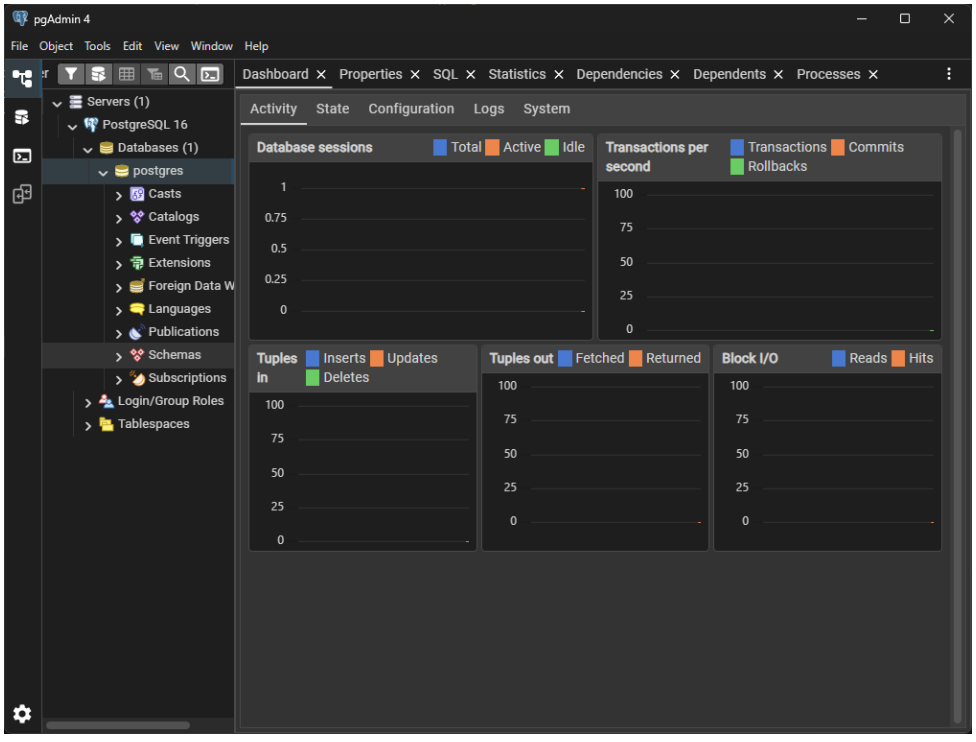
La guia indica el archivo equivocado, suponiendo que es envejemplo en vez:

```
copy envjemplo .env
```

Dentro del .env veo que necesita PostgreSQL, asi que lo instalo correctamente.
postgres postgres 5432
Postgres se rompe completamente y tengo que desinstalarlo entero y volver a instalarlo.
Creo una base de datos dentro de Postgres llamada postgres tal y como indica el ejercicio:



Creacion de base de datos con el mismo



Ahora edit el .env con mis datos:

```
# Reemplaza con tu conexion real a PostgreSQL
DATABASE_URL="postgresql://postgres:1BfagdRsdRa1mEaXpXJF@152
    .53.189.194:6183/postgres"
```

Por:

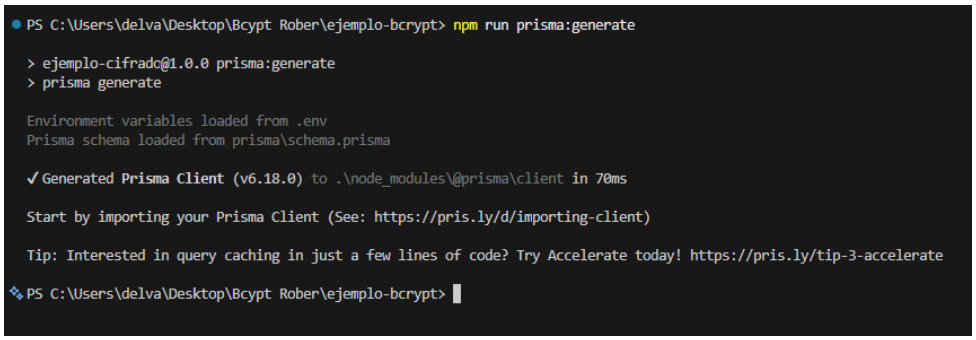
```
DATABASE_URL="postgresql://postgres:postgres@localhost:5432/
    postgres"
```

Cambio tambien el secret:

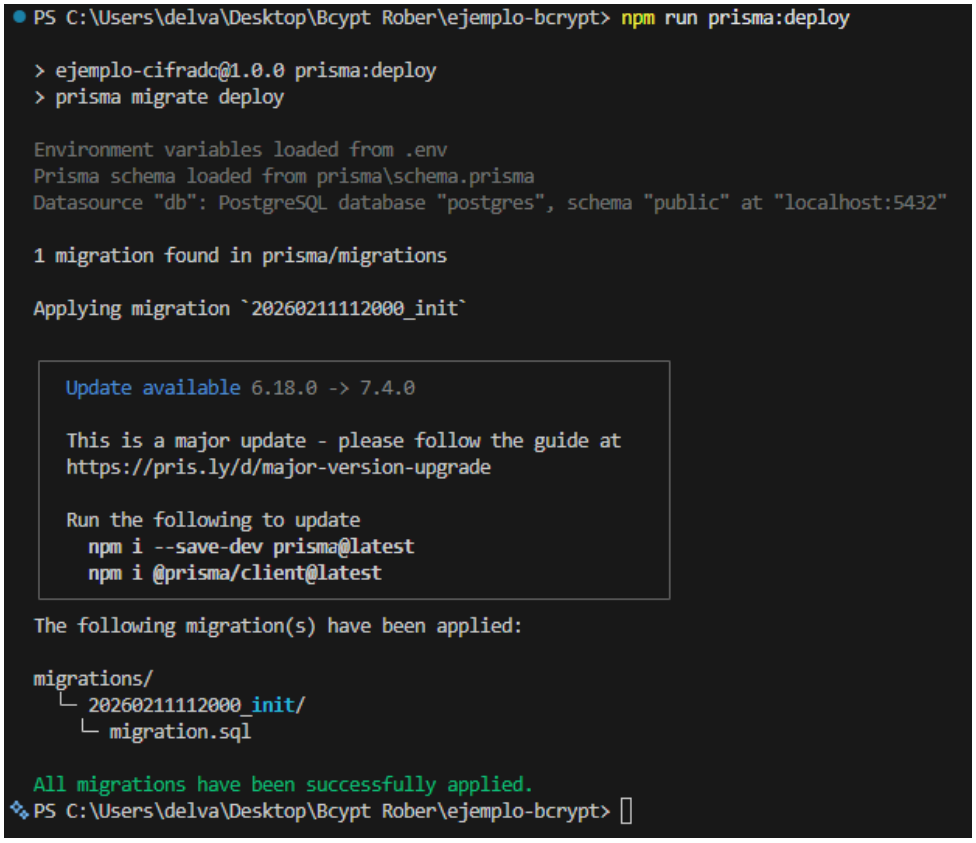
```
JWT_ACCESS_SECRET="postgrespostgres"
```

Ahora por fin puedo ejercutar el paso 4:

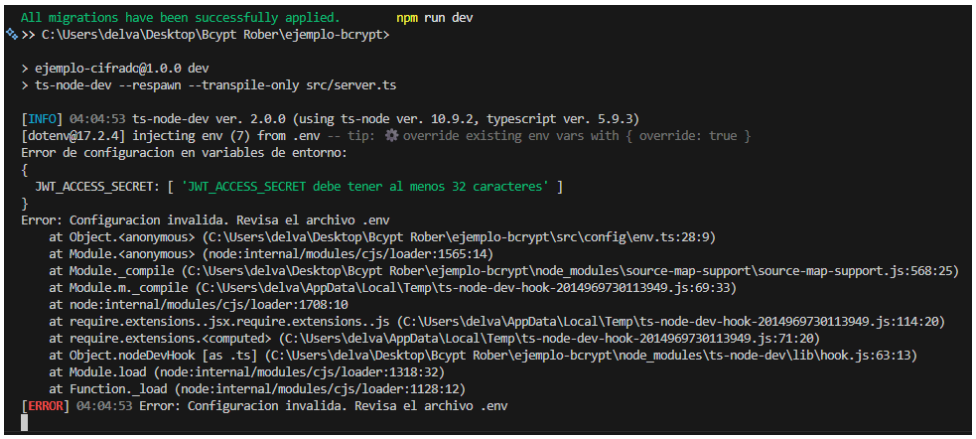
```
npm run prisma:generate
```




```
npm run prisma:deploy
```



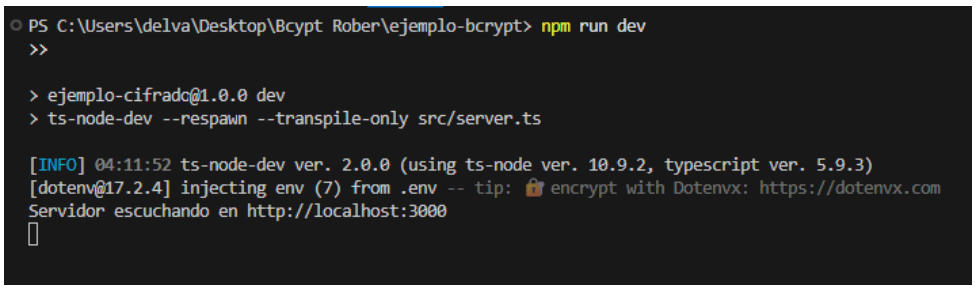

```
npm run dev
```



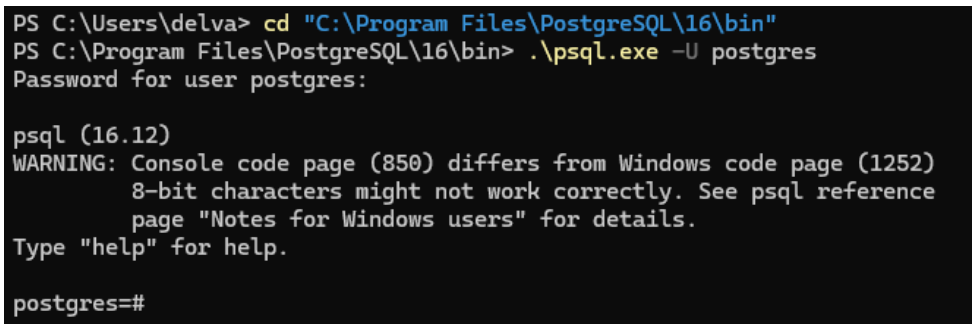
Da error debido a que la contraseña debe de tener mas de 32 caracteres. La cambio a:

JWT_ACCESS_SECRET=”
postgrespostgres12345678910111213141516171819202122232425”

Ahora ya funciona correctamente:



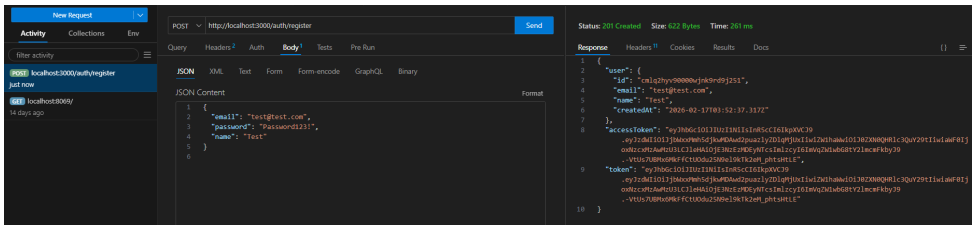
Pruebo que este funcionando Postgres correctamente:



Servidor no es capaz de leer el json:

```
PS C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt> npm run dev
  >>

  > ejemplo-cifrado@1.0.0 dev
  > ts-node-dev --respawn --transpile-only src/server.ts

  [INFO] 04:17:36 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 5.9.3)
  [dotenv@17.2.4] injecting env (7) from .env -- tip: add secrets lifecycle management: https://dotenvx.com/ops
  Servidor escuchando en http://localhost:3000
  SyntaxError: Expected property name or '}' in JSON at position 1 (line 1 column 2)
      at JSON.parse (<anonymous>)
      at parse (C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt\node_modules\body-parser\lib\types\json.js:72:19)
      at C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt\node_modules\body-parser\lib\read.js:162:18
      at AsyncResource.runInAsyncScope (node:async_hooks:211:14)
      at invokeCallback (C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt\node_modules\raw-body\index.js:238:16)
      at done (C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt\node_modules\raw-body\index.js:227:7)
      at IncomingMessage.onEnd (C:\Users\delva\Desktop\Bcypt Rober\ejemplo-bcrypt\node_modules\raw-body\index.js:287:7)
      at IncomingMessage.emit (node:events:524:28)
      at endReadableNT (node:internal/streams/readable:1698:12)
      at processTicksAndRejections (node:internal/process/task_queues:90:21) {
    expose: true,
    statusCode: 400,
    status: 400,
    body: '{\\',
    type: 'entity.parse.failed'
  }
```

Ahora usando thunder client creo una peticion con los datos de ejemplo:

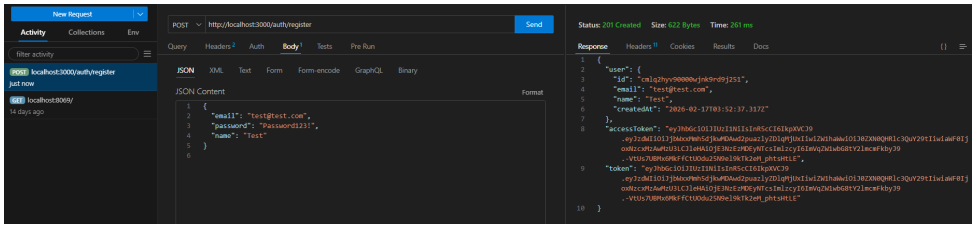


Funciona, está configurado correctamente.

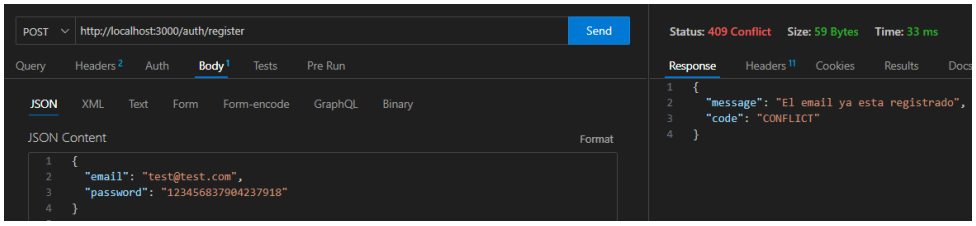
5. Escenarios de validacion manual

5.1. Registro valido y rechazo por email duplicado

Valido:
Registro de usuario: POST /auth/register

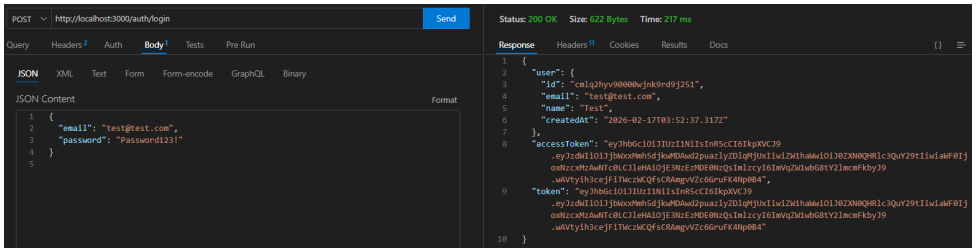


Invalido:



5.2. Login correcto e incorrecto

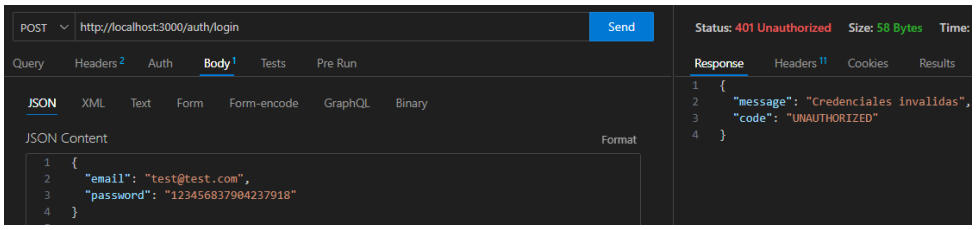
Correcto: Login: POST /auth/login



Podemos ver el token:

"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiJjbWxxMmh5djkwMDAwd2puazlyZDlqMjUxIiwiZW1haWwiOiJ
0ZXN0QHRlc3QuY29tIiwiaWF0IjoxNzcxMzAwNTc0
LCJleHAiOjE3NzEzMDE0NzQsImlzcyI6ImVqZW1wbG8tY2lmcmFkbyJ9.
wAVtyih3cejFiT
WczWCQfsCRAmgvVZc6GruFK4Np0B4",
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiJjbWxxMmh5djkwMDAwd 2
puazlyZDlqMjUxIiwiZW1haWwiOiJ0ZXN
0QHRlc3QuY29tIiwiaWF0IjoxNzcxMzAwN
Tc0LCJleHAiOjE3NzEzMDE0NzQsImlzcyI6ImVqZW1wbG8tY2lmcmFkbyJ9
.wAVtyih3cejFiTWczWCQfsCRAmgvVZc6GruFK4Np0B4"

Incorrecto:



5.3. Confirmar en DB que no hay passwords ni API keys en claro

Tras buscar la estructura adecuada, pude ver el hash, en este caso:

```
postgres=# \d "User"
                                Table "public.User"
    Column    |              Type              | Collation | Nullable |      Default
--------------+--------------------------------+-----------+----------+-------------------
 id           | text                           |           | not null |
 email        | text                           |           | not null |
 name         | text                           |           |          |
 passwordHash | text                           |           | not null |
 createdAt    | timestamp(3) without time zone |           | not null | CURRENT_TIMESTAMP
 updatedAt    | timestamp(3) without time zone |           | not null | CURRENT_TIMESTAMP
Indexes:
    "User_pkey" PRIMARY KEY, btree (id)
    "User_email_key" UNIQUE, btree (email)
Referenced by:
    TABLE ""ApiCredential"" CONSTRAINT "ApiCredential_userId_fkey" FOREIGN KEY ("userId") R
E CASCADE ON DELETE CASCADE


postgres=# SELECT email, "passwordHash" FROM "User";
     email     |                         passwordHash
---------------+--------------------------------------------------------------
 test@test.com | $2b$12$j1kShkDtzuq8Exs.Inz4uOECGnhz.lQy6TrcRnrT2TAvWAve8FYqi
(1 row)
```

Siendo el hash:

$2b$12$j1kShkDtzuq8Exs . Inz4uOECGnhz . lQy6TrcRnrT2TAvWAve8FYqi

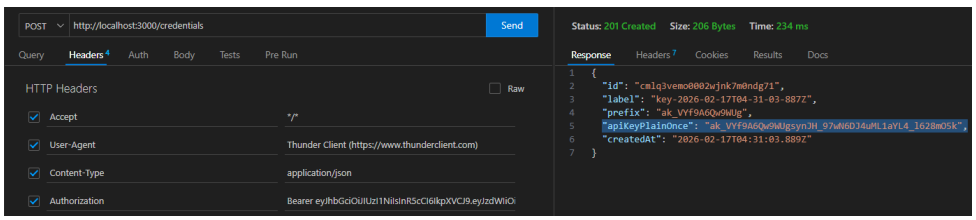
Para las api keys:

```
postgres=# \d "ApiCredential"
                            Table "public.ApiCredential"
   Column   |              Type              | Collation | Nullable |      Default
------------+--------------------------------+-----------+----------+-------------------
 id         | text                           |           | not null |
 userId     | text                           |           | not null |
 label      | text                           |           | not null |
 keyPrefix  | text                           |           | not null |
 keyHash    | text                           |           | not null |
 lastUsedAt | timestamp(3) without time zone |           |          |
 createdAt  | timestamp(3) without time zone |           | not null | CURRENT_TIMESTAMP
Indexes:
    "ApiCredential_pkey" PRIMARY KEY, btree (id)
    "ApiCredential_keyPrefix_idx" btree ("keyPrefix")
    "ApiCredential_userId_idx" btree ("userId")
Foreign-key constraints:
    "ApiCredential_userId_fkey" FOREIGN KEY ("userId") REFERENCES "User"(id) ON UPDATE C


postgres=# SELECT label, "keyHash" FROM "ApiCredential";
 label | keyHash
-------+---------
(0 rows)


postgres=# SELECT label, "hashedKey" FROM "ApiCredential";
```

Muestra que no hay debido a que no fue creada (siendo el siguiente paso)

5.4. Crear API key y comprobar que solo se muestra una vez

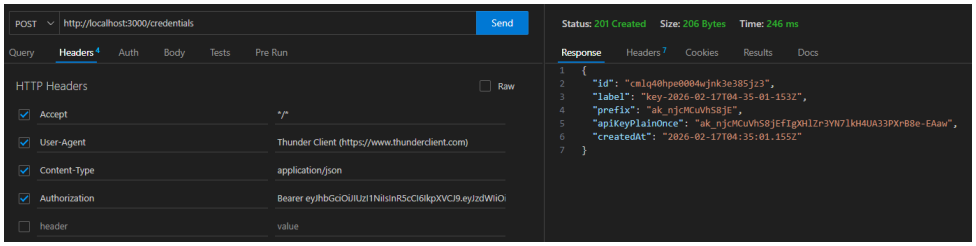
Para ello debemos de ir a headers Content type application/json y Authorization Bearer (accesToken)



Con esto obtenemos la API key:

"apiKeyPlainOnce": "ak_VYf9A6Qw9WUgsynJH_97wN6DJ4uML1aYL4_l628mO5k",

Probamos a pedirla de nuevo:

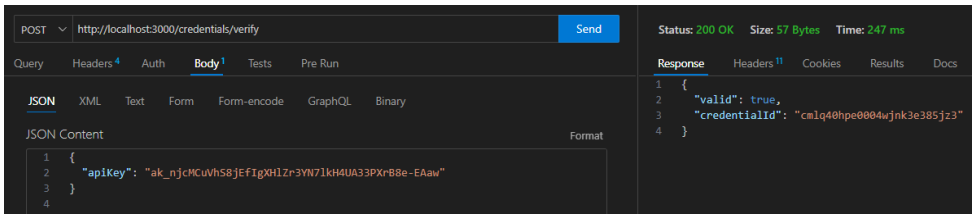


Mostrando una completamente nueva:

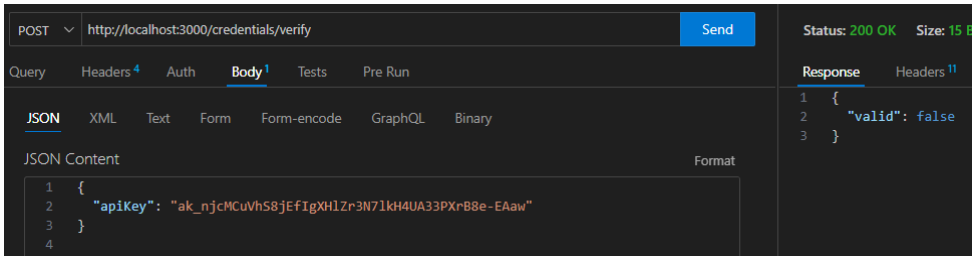
"apiKeyPlainOnce": "ak_njcMCuVhS8jEfIgXHlZr3YN7lkH4UA33PXrB8e-EAaw",

5.5. Verificar API key valida e invalida

Valida:



Invalida: (quitandole un numero cualquiera)



Con esto ya comprobé todas las funciones pedidas en el ejecicio de git.