

Documentación de la aplicación para el Alkemy Challenge

Nicolas Gallino

Fecha límite
12/04/2023

Cencosud

Tabla de contenido

<i>Instalación de la API para su utilización.....</i>	3
Opcional.	3
Iniciar el proyecto.	3
Iniciar base de datos.	3
Levantar la aplicación.....	3
<i>Funcionalidades.</i>	4
Reclamos.	7
Paginación	12
<i>Autorización.....</i>	13
<i>Carga de imágenes.....</i>	14
<i>AWS.</i>	19
<i>Notas</i>	20

Instalación de la API para su utilización

Opcional.

Crear una instancia EC2, instalar en ella Docker, AWS Cli, Visual Studio Code y Git.
Descargar el proyecto desde el bucket S3 de AWS a través del link proporcionado.
Descargarlo utilizando AWS Cli o conectándose remotamente a la EC3 y hacerlo desde la interfaz gráfica de Amazon.

Iniciar el proyecto.

Descomprimir el proyecto, abrir una terminal en la carpeta y ejecutar: 'yarn install'.
Abrir Visual Studio Code para visualizar la aplicación y en el archivo. env configurar las variables de entorno deseadas.

Iniciar base de datos.

La base de datos esta dockerizada por lo que para levantarla debemos ejecutara en el directorio del proyecto el comando 'docker-compose up' . Para esto es necesario tener Docker previamente instalado y corriendo.

Levantar la aplicación.

Para inicializar la aplicación ejecutamos el comando 'yarn start:dev' para activar el modo -watch de la ejecución.

A este punto tanto la aplicación como la base de datos, están correctamente funcionando.
Podemos encontrar la API en: <http://localhost:3000/graphql>

Funcionalidades.

La API cuenta con 4 modulos, 3 de ellos (Auth, Users, Reclamos), son GRAPHQL, por lo que para utilizarlos se tienen que emplear Querys y Mutations. En cuanto al módulo 'files', utilizamos los métodos comunes de REST para permitir la carga de imágenes.

Primero, necesitamos registrar al menos un usuario. Utilizaremos la mutation 'signup'.

```
@Resolver( () => AuthResponse )
export class AuthResolver {

    constructor(
        private readonly authService: AuthService
    ) {}

    @Mutation( () => AuthResponse , { name: 'signup' })
    async signup(
        @Args('signupInput') signupInput: SignupInput
    ): Promise<AuthResponse> {
        return this.authService.signup( signupInput );
    }

    @Mutation( () => AuthResponse , { name: 'login' })
    async login(
        @Args('loginInput') loginInput: LoginInput
    ): Promise<AuthResponse> {
        return this.authService.login( loginInput );
    }
}
```

The screenshot shows the GraphQL playground interface with the 'Mutation' tab selected. In the 'Operation' section, the 'signup' mutation is selected. The 'Variables' panel contains the following JSON:

```
{
  "signupInput": {
    "email": "alguien@alguien.com",
    "password": "Password1",
    "fullName": "Alguien Diaz"
  }
}
```

The 'Result' panel displays the response from the server:

```
data: {
  "signup": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ijg4NDJhODI2LTU3NjYtNDMzN105ZDQ1LTcwYzc5OWY0YWZlYiIsInlhC16HTY4MDU0MjUSN1w1ZxhW1oxNjgwNTU2OTk2fQ.4H0cm-w5UXdpkBh8wN8xYfmLA953Tmj71lfbOccJnf4"
  }
}
```

```
        {
      "signupInput": {
        "email": "alguien@alguien.com",
        "password": "Password1",
        "fullName": "Alguien Diaz"
      }
    }
```

En la base de datos podemos corroborar que el usuario se creó correctamente y además su contraseña se guardó encriptada con BYCRYPT.

A continuación, iniciaremos sesión con el mismo email y password ingresados anteriormente y guardaremos el token que nos devuelve para poder autenticarnos y autorizarnos.

The screenshot shows a GraphQL mutation interface. At the top, there's a header with "Mutation" and a "+" button. Below it, the "Operation" section contains the following GraphQL code:

```
1 mutation Mutation($loginInput: LoginInput!) {
2   login(loginInput: $loginInput) {
3     token
4   }
5 }
```

To the right of the code, there are three small icons: a file, a clipboard, and a copy symbol. Next to them is the text "STATUS 200 | 136ms | 219B".

Below the operation, there are tabs for "Variables", "Headers", "Script", and "NEW!". The "Variables" tab is selected, showing the following JSON input:

```
1 {
2   "loginInput": {
3     "email": "alguien@alguien.com",
4     "password": "Password1",
5   }
6 }
```

On the far right of the variables panel, there's a "JSON" button with a dropdown arrow.

On the right side of the interface, there's a large text area showing the response from the server. It starts with "data": { and ends with "}".

```
[{"data": {"login": {"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ijg4NDJiMDI2LTU3NjYtNDMzMj05ZDQ1LTcwYzc5OWY0YmZlYiIsImhdCI6MTY4MDU0MjkyMiwiZXhwIjoxNjgwNTU3MzIyfQ.WYVAO2SdnCPPEQJITm7J10PWGodLbMAjlruRCNdtyk"}}}
```

Iniciamos sesión satisfactoriamente.

Reclamos.

The screenshot shows a GraphQL mutation editor interface. At the top, there's a header bar with icons for download, save, and a blue button labeled "Mutation". Below the header, the "Operation" tab is active, displaying the following GraphQL mutation code:

```
1 mutation Mutation($createReclamoInput: CreateReclamoInput!) {  
2   createReclamo(createReclamoInput: $createReclamoInput) {  
3     id  
4     titulo  
5     numerodereclamo  
6     descripcion  
7     detalleDeCompra {  
8       id  
9       fecha  
10      factura  
11      codProd  
12    }  
13  }  
14}  
15
```

Below the code, there are three tabs: "Variables", "Headers", and "Script". The "Variables" tab is selected and contains the following JSON input:

```
1 {  
2   "createReclamoInput": {  
3     "titulo": "Primer reclamo",  
4     "numerodereclamo": 1,  
5     "descripcion": "Reclamo de prueba",  
6     "detalledecompra": {  
7       "codProd": "1",  
8       "factura": 1,  
9       "fecha": "03-04-2023"  
10     }  
11   }  
12 }
```

On the right side of the "Variables" tab, there's a "JSON" button. The entire interface has a dark theme with light-colored text and code highlighting.

```
{  
  "createReclamoInput": {  
    "titulo": "Primer reclamo",  
    "numerodereclamo": 1,  
    "descripcion": "Reclamo de prueba",  
    "detailedecompra": {  
      "codProd": "1",  
      "factura": 1,  
      "fecha": "03-04-2023"  
    }  
  }  
}
```

The screenshot shows a GraphQL playground interface. At the top, there is a code editor containing a mutation query:

```
1  mutation Mutation($createReclamoInput: CreateReclamoInput!) {  
2    createReclamo(createReclamoInput: $createReclamoInput) {  
3      id  
4      titulo  
5      numerodereclamo  
6      descripcion  
7      detalleDeCompra []  
8        | id  
9        | fecha  
10       | factura  
11       | codProd  
12     | ]  
13   }  
14 }  
15
```

Below the code editor, there are tabs for "Variables", "Headers", and "Script". The "Headers" tab is selected, showing a dropdown menu with "Authorization" and a value input field containing "Bearer eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ". There is also a trash icon next to the value input field.

El token lo ingresamos como Header, siguiendo a la palabra Bearer y un espacio.

	id	titulo	numerodereclamo	descripcion	userid	detalleDeComprad
	f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba	dasdas	3	dsa	e1507003-ffff-4604-9f73-fe645f89a85c	e1507003-ffff-4604-9f73-fe645f89a85c

En la base de datos se guarda correctamente el reclamo, y su CSV relacionado. Además, el reclamo almacena que usuario lo hizo.

Todas las Querys funcionan correctamente.

Traer todos los reclamos.

```

query Query {
  reclamos {
    id
    titulo
    numerodereclamo
    descripcion
    detalleDeCompra {
      id
      factura
      codProd
      fecha
    }
  }
}

```

Variables Headers Script NEW!

JSON

```

{
  "data": {
    "reclamos": [
      {
        "id": "f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba",
        "titulo": "dasdas",
        "numerodereclamo": 3,
        "descripcion": "dsa",
        "detalleDeCompra": {
          "id": "e1507003-ffff-4604-9f73-fe645f89a85c",
          "factura": 321,
          "codProd": "qsdf",
          "fecha": "Dsadsa"
        }
      }
    ]
  }
}

```

Traer un reclamo dado su título.

The screenshot shows a GraphQL playground interface. The top bar includes a 'Query' button and status information: STATUS 200 | 12.0ms | 239B. The main area has tabs for 'Operation', 'Variables', 'Headers', and 'Script'. The 'Operation' tab contains the following query:

```
1 query Query($search: String) {
2   reclamos(search: $search) {
3     id
4     titulo
5     numerodereclamo
6     descripcion
7     detalleDeCompra []
8     id
9     factura
10    codProd
11    fecha
12  }
13}
```

The 'Variables' tab shows:

```
1 {
2   "search": "dasdas"
3 }
```

The right panel displays the response in JSON format:

```
{
  "data": {
    "reclamos": [
      {
        "id": "f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba",
        "titulo": "dasdas",
        "numerodereclamo": 3,
        "descripcion": "dsas",
        "detalleDeCompra": [
          {
            "id": "e1507003-ffffb-4604-9f73-fe645f89a85c",
            "factura": 321,
            "codProd": "qsd",
            "fecha": "Dsadas"
          }
        ]
      }
    ]
  }
}
```

Traer un reclamo dado su ID.

The screenshot shows a GraphQL playground interface. The top bar includes a 'Query' button and status information: STATUS 200 | 23.0ms | 233B. The main area has tabs for 'Mutation', 'Query', and '+'. The 'Mutation' tab contains the following query:

```
1 query Query($itemId: ID!) {
2   item(id: $itemId) {
3     id
4     titulo
5     numerodereclamo
6     descripcion
7     detalleDeCompra []
8     id
9     factura
10    codProd
11    fecha
12  }
13}
```

The 'Variables' tab shows:

```
1 {
2   "itemId": "f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba"
3 }
```

The right panel displays the response in JSON format:

```
{
  "data": {
    "item": {
      "id": "f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba",
      "titulo": "dasdas",
      "numerodereclamo": 3,
      "descripcion": "dsas",
      "detalleDeCompra": [
        {
          "id": "e1507003-ffffb-4604-9f73-fe645f89a85c",
          "factura": 321,
          "codProd": "qsd",
          "fecha": "Dsadas"
        }
      ]
    }
  }
}
```

Modificar un reclamo.

The screenshot shows the GraphQL playground interface with the following details:

- Documentation:** Root → Mutation → updateItem
- Operation:** mutation UpdateItem(\$updateItemInput: UpdateReclamoInput!) { updateItem(updateItemInput: \$updateItemInput) { id titulo numerodereclamo descripcion detalleDeCompra { id factura codProd fecha } } }
- Variables:** JSON input:

```
1 {  
2   "updateItemInput": {  
3     "id": "f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba",  
4     "descripcion": "Reclamo modificado"  
5   }  
6 }
```
- Response:** STATUS 200 | 45.0ms | 2548 bytes. The response shows the updated claim data with the new description.

Eliminar un reclamo.

The screenshot shows the GraphQL playground interface with the following details:

- Documentation:** Root → Mutation → removeltem → detalleDeCompra
- Operation:** mutation Mutation(\$removeItemId: ID!) { removeItem(id: \$removeItemId) { id titulo numerodereclamo descripcion detalleDeCompra { id factura codProd fecha } } }
- Variables:** JSON input:

```
1 {  
2   "removeItemId": "f13bdbbd-18f4-43a8-be6b-ca2a9aa24bba"  
3 }
```
- Response:** STATUS 200 | 85.0ms | 2548 bytes. The response shows the removed item data.

	id	titulo	numerodereclamo	descripcion	usuarioid	detalleDeCompraid
1	1	Primer reclamo	1	Este es el primer reclamo.	1	Detalle de compra 1
2	2	Segundo reclamo	2	Este es el segundo reclamo.	2	Detalle de compra 2
3	3	Tercer reclamo	3	Este es el tercer reclamo.	3	Detalle de compra 3
4	4	Cuarto reclamo	4	Este es el cuarto reclamo.	4	Detalle de compra 4
5	5	Quinto reclamo	5	Este es el quinto reclamo.	5	Detalle de compra 5
6	6	Sextavo reclamo	6	Este es el sextavo reclamo.	6	Detalle de compra 6
7	7	Séptimo reclamo	7	Este es el séptimo reclamo.	7	Detalle de compra 7
8	8	Octavo reclamo	8	Este es el octavo reclamo.	8	Detalle de compra 8
9	9	Nº9 reclamo	9	Este es el nº9 reclamo.	9	Detalle de compra 9
10	10	Nº10 reclamo	10	Este es el nº10 reclamo.	10	Detalle de compra 10

Paginación

Utilizamos la Paginación utilizando los argumentos de offset y limite. Estos se encargan de seleccionar cuales y el límite de reclamos a traer relativamente.

Operation

```

query($offset: Int, $limit: Int){
  reclamos(offset: $offset, limit: $limit) {
    id
    titulo
    numerodereclamo
    descripcion
    detalleDeCompra {
      id
      factura
      codProd
      fecha
    }
  }
}

```

Root > Query > reclamos

← **reclamos: [Reclamo!]!** ✓

Arguments

- ✓ offset: Int = 0
- ✓ limit: Int = 10
- + search: String

Fields ↓ ✓ ▾

- ✓ descripcion: String!
- ✓ detalleDeCompra: DetalleDeCompra!
- ✓ id: ID!
- ✓ numerodereclamo: Float!
- ✓ titulo: String!

Variables Headers Script NEW!

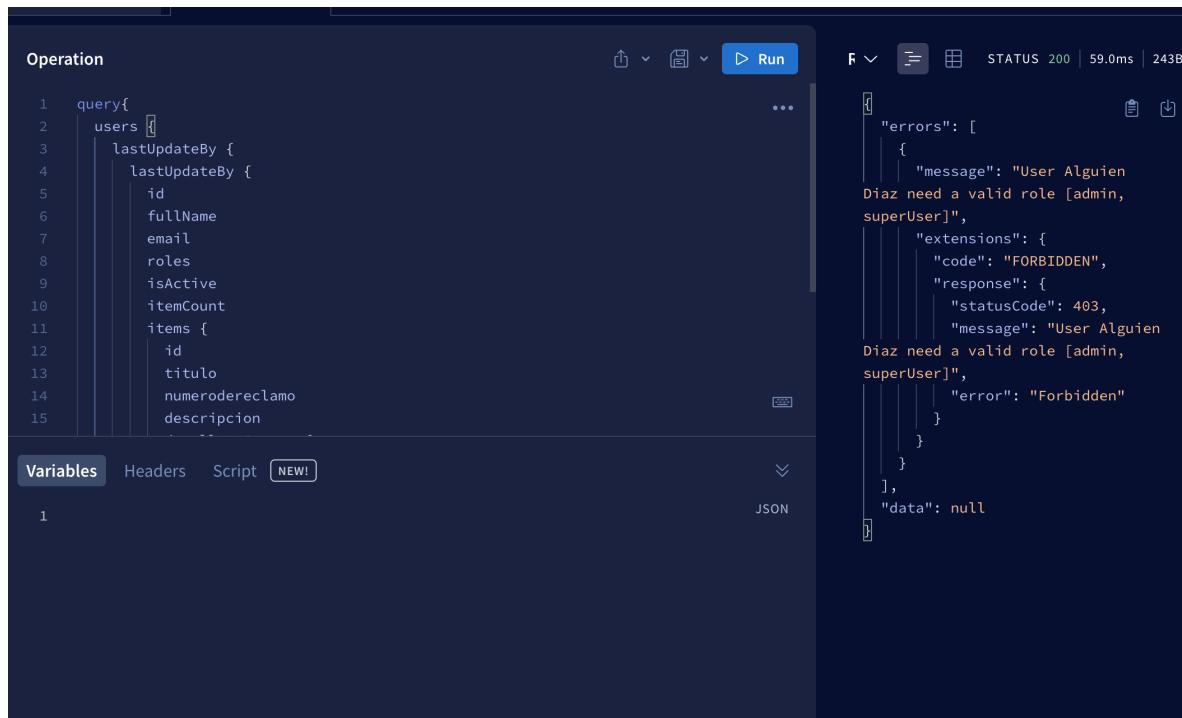
```

{
  "offset": null,
  "limit": null
}

```

Autorización

Un usuario con el rol de Admin, puede ver la información de los demás usuarios. En este caso se muestra como un usuario sin este rol no puede acceder a esta información corroborando que tanto la Autorización como Autentificación funcionan correctamente.



The screenshot shows a GraphQL playground interface. On the left, there is a code editor window titled "Operation" containing the following GraphQL query:

```
query{  users {    lastUpdateBy {      id      fullName      email      roles      isActive      itemCount      items {        id        titulo        numerodereclamo        descripcion      }    }  } }
```

Below the code editor are tabs for "Variables", "Headers", "Script", and "NEW!". The "Variables" tab is selected. To the right of the code editor is a "Run" button. The status bar at the top right indicates "STATUS 200 | 59.0ms | 243B".

The main panel on the right displays the JSON response. The response has an "errors" field containing an array of errors. Each error object has a "message" field with the text "User Alguien Diaz need a valid role [admin, superUser]", a "code" field with "FORBIDDEN", a "response" field with a "statusCode" of 403 and a "message" of "User Alguien Diaz need a valid role [admin, superUser]", and an "error" field with the value "Forbidden". The "data" field is null.

```
{"errors": [ { "message": "User Alguien Diaz need a valid role [admin, superUser]", "code": "FORBIDDEN", "response": { "statusCode": 403, "message": "User Alguien Diaz need a valid role [admin, superUser]" }, "error": "Forbidden" } ], "data": null}
```

Los usuarios se crean de manera automática con el rol 'user'. Por seguridad, para obtener el rol 'Admin' es necesario tener acceso a la base de datos y hacer la modificación correspondiente de forma directa.

Carga de imágenes.

Para subir imágenes utilizamos Postman y el método Post.

La carga, verifica el formato del archivo para impedir archivos maliciosos y su peso. En este ejemplo el archivo es muy pesado.

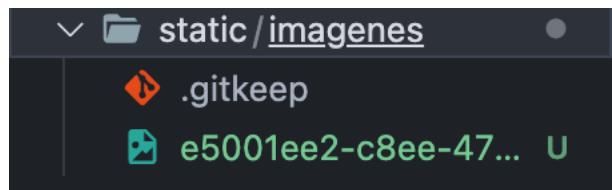
The screenshot shows a Postman request configuration for a POST method to the endpoint `localhost:3000/files/imagen`. The `Body` tab is selected, showing a single form-data entry named `Imagen` with the value set to `MicrosoftTeams-image.png`. Below the request details, the response section displays the following JSON:

```
1  {
2     "statusCode": 413,
3     "message": "File too large",
4     "error": "Payload Too Large"
5 }
```

The status bar at the bottom indicates a `413 Payload Too Large` error with a size of `323 B`.

The screenshot shows a POST request to `localhost:3000/files/imagen`. In the Body tab, there is a single form-data entry named `Imagen` with the value `MicrosoftTeams-image.png`. The response tab shows a `201 Created` status with a JSON body containing the key `fieldName` with the value `"MicrosoftTeams-image.png"`.

A la imagen subida se le asigna un UUID para identificarla y relacionarla con los reclamos.



Límite de tamaño (en bytes)

```

@Post('imagen')
@UseInterceptors(FileInterceptor('imagen', {
  fileFilter: fileFilter,
  limits: { fileSize: 3000000 },
  storage: diskStorage ({
    destination: './static/imagenes',
    filename: fileName
  })
}))
uploadImage(
  @UploadedFile() file: Express.Multer.File
){

  return { fieldName: file.originalname };

}

```

Verificación de formato.

```
export const fileFilter = (req: Express.Request, file: Express.Multer.File, callback: Function ) => {

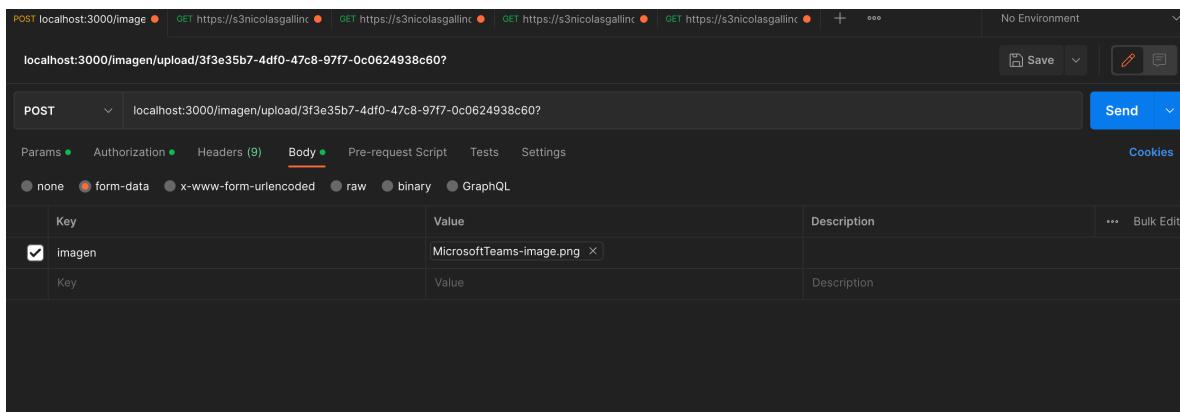
    if (!file) return callback ( new Error ('Image is empty'), false)

    const imageExtension = file.mimetype.split('/')[1];
    const validExtensions= ['jpg', 'jpeg', 'png', 'gif'];

    if (validExtensions.includes ( imageExtension )){

        return callback ( null, true )
    }

    callback (null, true);
}
```



The screenshot shows the Postman interface with a POST request to `localhost:3000/imagen/upload/3f3e35b7-4df0-47c8-97f7-0c0624938c60?`. The Body tab is selected, showing a form-data key `Imagen` with the value `MicrosoftTeams-image.png`.

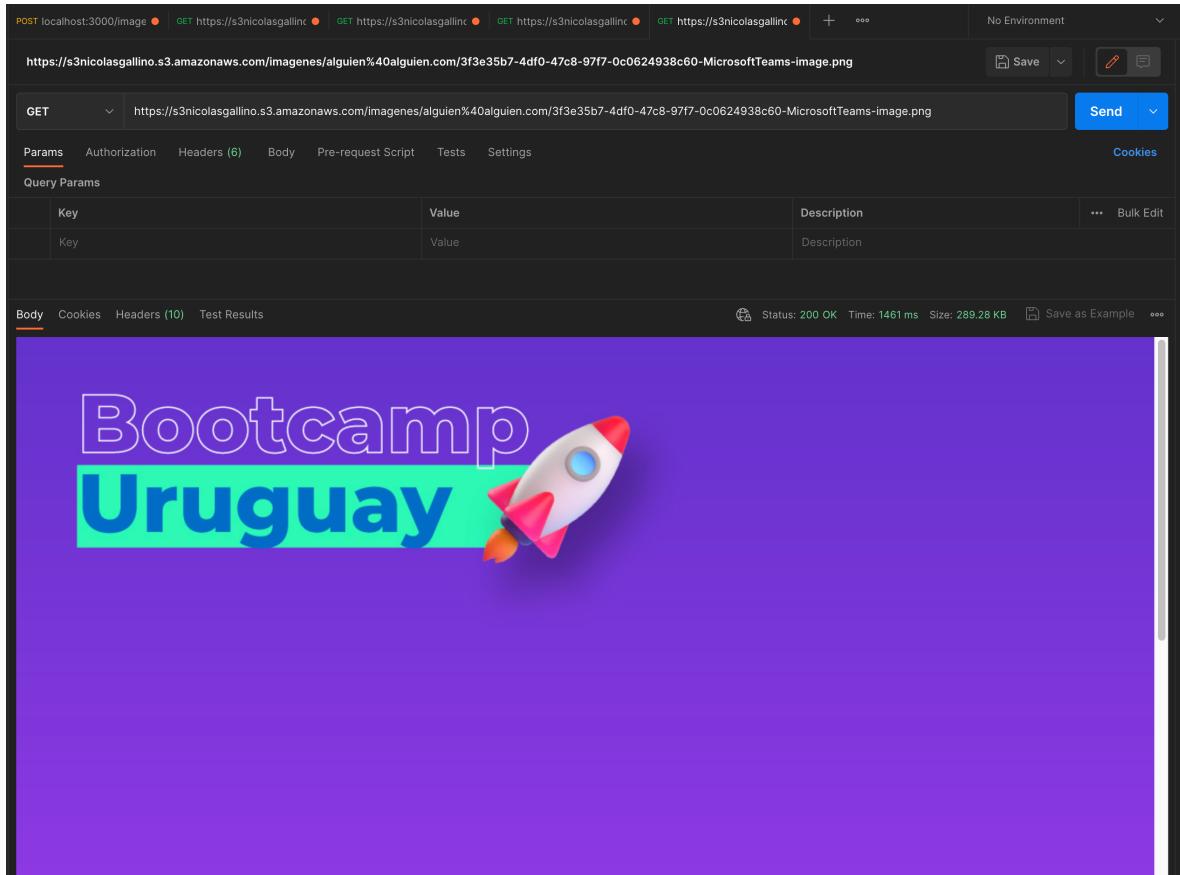
La imagen se carga a un reclamo, ingresando la id de este como parámetro.

The screenshot shows the Amazon S3 console interface. The path is: Amazon S3 > Buckets > s3nicolasgallino > imágenes > alguien@alguien.com/. The object name is alguien@alguien.com/. There is a button labeled 'Copiar URI de S3'. Below the object name, there are tabs for 'Objetos' (selected) and 'Propiedades'. A search bar says 'Buscar objetos por prefijo'. The table below lists one object:

	Nombre	Tipo	Última modificación	Tamaño	Clase de almacenamiento
<input type="checkbox"/>	3f5e35b7-4df0-47c8-97f7-0c0624938c60-MicrosoftTeams-image.png	png	5 Apr 2023 4:58:30 PM -03	288.9 KB	Estándar

Estas se suben a un bucket. Están dentro de una carpeta ‘imágenes’ y a su vez dentro de esta, en subcarpetas que se crean automáticamente con el email del usuario que creo el reclamo.

Al momento de subir la imagen, el api devuelve el link de la misma, a su vez queda guardada en la base de datos en la tabla de reclamos.
Haciendo GET a este link podemos previsualizar la imagen.



LOCAL PostgreSQL 14.4 : : AnyList : public.reclamos						
id	título	numerodereclamo	descripcion	usuarioid	detalleDeCompraid	imagen
3f3e35b7-4df0-47c8-97f7-0c0624938c60	ImagenTest	20	ImagenTest	8842b026-5766-4336-9d45-70c799f4afeb	f08586b7-1bda-4460-bc6... →	https://s3nicolasgallino.s3.amazonaws.com/imagenes/alguien%40alguien.com/3f3e35b7-4df0-47c8-97f7-0c0624938c60-MicrosoftTeams-image.png

Importante.

Es importante tener en cuenta que para la carga de imágenes es necesario estar autenticado. Esto se puede hacer obteniendo el token iniciando sesión utilizando las querys Graph y utilizando ese mismo token en Postman.

AWS.

La aplicación se puede encontrar en un Bucket S3 de AWS.

The screenshot shows the AWS S3 console interface for the bucket 's3nicolasgallino'. The bucket is set to 'Accesible públicamente' (Publicly Accessible). The 'Objetos (1)' section displays a single item: 'alkemychallenge-master/' which is a 'Carpeta' (Folder). Below the table are standard S3 actions: Copiar URI de S3, Copiar URL, Descargar, Abrir, Eliminar, Acciones, and Crear carpeta. A search bar and navigation controls are also present.

```
src > files > files.service.ts > ...
1 import { Injectable } from '@nestjs/common';
2 import { S3 } from 'aws-sdk';
3 import { ReclamosService } from 'src/reclamos/reclamos.service';
4 import { User } from 'src/users/entities/user.entity';
5
6
7 @Injectable()
8 export class FilesService {
9
10   constructor(private readonly reclamoService: ReclamosService) {}
11
12   async upload(reclamoId: string, user:User, file: Express.Multer.File){
13     const s3 = new S3();
14     const uploadParams = {
15       Bucket:'s3nicolasgallino',
16       Body: file.buffer,
17       Key:`imagenes/${user.email}/${reclamoId}-${file.originalname}`,
18       ContentType: file.mimetype
19     }
20
21     const img = await s3.upload(uploadParams).promise();
22
23     await this.reclamoService.addImage (reclamoId, user, img.Location);
24
25     return img.Location;
26   }
27 }
```

Notas

En el código del servicio encargado de la subida de las imágenes podemos ver la propiedad ‘Bucket’ en la que tenemos que ingresar el nombre de un Bucket ya creado. Las credenciales de AWS se ingresan en el archivo ‘credentials’ dentro de la carpeta oculta en la raíz ‘.aws’. Debido a que para este proyecto utilice una cuenta AWS Academy, las keys varían por lo que es recomendado utilizar un bucket propio o de lo contrario contactarse conmigo al momento de la corrección para brindar las keys correspondientes.

Para monitorear la base de datos, la cual es una imagen de Postgres desplegada utilizando Docker, use Tableplus.

Para el control de versiones, Git y GitHub donde también se puede encontrar el repositorio.

Dentro del proyecto podemos encontrar un archivo ‘README’ el cual también cuenta con información útil para la utilización de la aplicación.

FIN DE DOCUMENTACION.