

PROYECTO 02

INTEGRACIÓN DE APLICACIONES

Autor : José Alejandro Pastor Mayor.

Asignatura : Integración de Aplicaciones.

Fecha Entrega: 7/01/2025

Índice del Documento:

☒ Laboratorio 1

- ~~1. Introducción.~~
- ~~2. Tecnologías Utilizadas.~~
- ~~3. Funcionalidades del proyecto.~~
- ~~4. Comandos.~~
- ~~5. Conclusiones.~~

☐ Laboratorio 2

1. Seguimiento del Proceso.
2. Tecnologías Utilizadas.
3. Nuevas funcionalidades y archivos.
4. Pequeño avance visual.
5. Conclusiones.

Lab 1 : Introducción: Twitter Lite

Este proyecto tiene como objetivo desarrollar una versión simplificada de Twitter denominada Twitter Lite, implementada en JavaScript utilizando el entorno de desarrollo Visual Studio Code y Node.js como tecnología base. A través de esta iniciativa, se busca introducir a los participantes en la arquitectura de aplicaciones desacopladas y la integración mediante el uso de un almacén de datos compartido basado en MongoDB.

Twitter Lite permite a los usuarios realizar operaciones clave como:

- Registro y autenticación.
- Publicación y exploración de mensajes (tweets).
- Gestión de relaciones de seguidores (follow/unfollow).
- Interacción con mensajes mediante likes y dislikes.

El proyecto adopta una arquitectura modular con componentes claramente definidos:

1. Interfaz de línea de comandos (CLI): Facilita la interacción del usuario mediante comandos directos.
2. API: Define las operaciones esenciales, asegurando una separación efectiva entre el front-end y el back-end.
3. Modelo de datos: Basado en colecciones de usuarios y tweets en MongoDB, soporta relaciones y referencias para operaciones como retweets y seguidores.

Propósitos Clave

1. Explorar el diseño de aplicaciones escalables y reutilizables.
2. Implementar prácticas de autenticación segura y gestión de datos.
3. Facilitar consultas avanzadas y operaciones eficientes en bases de datos.



MongoDB es una base de datos NoSQL orientada a documentos. A diferencia de las bases de datos relacionales tradicionales, almacena los datos en un formato flexible de tipo JSON (BSON internamente), lo que facilita la representación de estructuras complejas y permite escalar horizontalmente. En el contexto de este proyecto, **MongoDB** actúa como el almacén de datos principal para gestionar usuarios, tweets, relaciones de seguimiento y preferencias (likes & dislikes).



Node.js es un entorno de ejecución de **JavaScript** que permite desarrollar aplicaciones del lado del servidor. Se destaca por su arquitectura no bloqueante y basada en eventos, lo que lo hace ideal para aplicaciones en tiempo real y de alta concurrencia. En este proyecto, utilizamos Node.js como base para implementar la lógica de negocio y la interfaz de la aplicación **Twitter Lite**.

Lab 1 : Funcionalidades del Proyecto

A Nivel de Programación

- Control de Salida: Posibilidad de salir de la aplicación fácilmente desde el menú o utilizando el comando logout, que además cierra la sesión activa del usuario.
- Gestión de Errores:
 - Los errores de cada comando han sido revisados y controlados con mensajes claros y personalizados.
 - Solución a problemas relacionados con el manejo de espacios mediante el uso de match en lugar de split (Minimist).
 - Parches para evitar errores al repetir parámetros en Minimist, que anteriormente generaban arrays no deseados.
- Sistema de Ayuda: Cada comando cuenta con una opción **--help**, detallada y bien explicada, para guiar al usuario.
- Registro de Actividades: Implementación de un log de mensajes para crear una pila de registros, con la posibilidad de guardar dichos logs.
- Consultas Optimizadas:
 - Queries bien diseñadas y funcionales para los comandos de listado.
 - Uso de referencias por ID de objetos MongoDB para búsquedas eficientes (e.g., Follow/Unfollow).
- Gestión de Usuarios y Actualizaciones:
 - Actualización de parámetros individuales (e.g., email, nick), evitando modificaciones masivas.
 - Chequeo exhaustivo durante las actualizaciones: se cancelan si el email o el nick ya están en uso.
 - Cambios dinámicos en el prompt según el estado del usuario (e.g., login o actualización de nick).
- Gestión de Tweets y Relaciones:
 - Inclusión de un campo Owner en los documentos de tweets, con información del creador para una mejor gestión.
 - Like y Dislike alternados: un like elimina un dislike previo, y viceversa.
 - Validaciones en likes/dislikes para evitar autoevaluaciones o acciones repetidas.
- Autocompletado Inteligente: Implementación de autocompletado para comandos y sugerencias útiles.

A Nivel Visual

- Diseño y Organización:
 - Interfaz elegante con colores llamativos y una estructura clara para evitar confusiones.
 - Separación de menús según el contexto (e.g., menú de Login y menú Principal).
- Mensajes Personalizados:
 - Mensajes de bienvenida y despedida para mejorar la experiencia del usuario.
- Facilidad de Navegación: Menús y comandos diseñados para una interacción intuitiva y sin complicaciones.

Interfaz de Entrada:

- Login
- Añadir Usuario

```
=====
                        LOGIN MENÚ
=====

1. login -e <email> -p <password>
2. addUser -n <name> -s <surname> -e <email> -p <password> -i <nick>
3. exit

Haz uso de los comandos anteriores para iniciar sesión o registrarse.
Para más información acerca de un comando utiliza <cmd> --help

[TW Lite] : 
```

Parámetro opcional para ayuda **--help** para login y addUser:

```
[TW Lite] : login --help

=====
                        MENÚ DE AYUDA : COMANDO LOGIN
=====

• Descripción: Comando de autenticación para un usuario.
• Uso: login -e <email> -p <password>.
• Variables:
    |-e| : <email>      -> Correo electrónico asociado a la cuenta de usuario.
    |-p| : <password>   -> Contraseña registrada para el usuario.

NOTA: La ejecución del comando no está disponible una vez autenticado.
```

```
[TW Lite] : addUser --help

=====
                        MENÚ DE AYUDA : COMANDO ADDUSER
=====

• Descripción: Registrar a un nuevo usuario en la base de datos.
• Uso: addUser -n <name> -s <surname> -e <email> o <password> -i <nick>.
• Variables:
    |-n| : <name>       -> Nombre del usuario que se está registrando.
    |-s| : <surname>    -> Primer apellido del usuario.
    |-e| : <email>      -> Correo electrónico vinculado a la cuenta de usuario.
    |-p| : <password>   -> Contraseña deseada para el usuario.
    |-i| : <nick>       -> Apodo o alias deseado dentro de la aplicación.

• Requisitos: Todos los parámetros son necesarios para la creación de un nuevo usuario.

NOTA: La ejecución del comando no está disponible una vez autenticado.
```

Accedemos a la aplicación / **Menú:**

```
[TW Lite] : login -e admin -p admin
[LOG] Acaba de acceder a la plataforma el usuario <ADMIN> con email: <admin>
[Éxito] >> Bienvenido a Twitter ADMIN. ¿Qué tal si empezamos a Twittear?

=====
                        MENÚ PRINCIPAL
=====

..... COMANDOS PARA LOS USUARIOS .....

1. listUsers -q <query> -ini <ini> -count <count> -sort <sort>
2. updateUser -n <name> -s <surname> -e <email> -p <password> -i <nick>
3. listFollowing -q <query> -ini <ini> -count <count> -sort <sort>
4. listFollowers -q <query> -ini <ini> -count <count> -sort <sort>
5. follow --id <userID>
6. unfollow --id <userID>

..... COMANDOS PARA LOS TWEETS .....

7. addTweet -c <content>
8. addRetweet --id <tweetID>
9. listTweets -q <query> -ini <ini> -count <count> -sort <sort>
10. like --id <tweetID>
11. dislike --id <tweetID>

12. logout

Para más información acerca de un comando
Haz uso de <cmd> --help

ADMIN : █
```

Cada comando dispone de su propio menú de ayuda con **-help**.
Mostramos alguno a continuación:

```
ADMIN : listUsers --help

=====
                        MENÚ DE AYUDA : COMANDO LISTUSERS
=====

• Descripción: Listar a los usuarios registrados en la aplicación.
• Uso: listUsers -q <query> -i <ini> -c <count> -s <sort>.
• Variables:
  |-q| : <query>      -> Especifica una consulta.
  |-i| : <ini>        -> Índice del primer resultado mostrado.
  |-c| : <count>      -> Indica el número máximo de resultados mostrados.
  |-s| : <sort>       -> Ordenar los resultados por +|- campos.

• Especificaciones:
  - El uso del comando no va ligado a sus variables. Se puede ejecutar con o sin ellas.
• Ejemplo de uso:
  - listUsers -q '{ name : "alex" }' -c 1

NOTA: La ejecución del comando solamente estará disponible una vez autenticado.
```

Funcionalidad de los Comandos:

Listar Usuarios (Sin query):

```
ADMIN : listUsers
```

| (index) | id | name | surname | email | nick | following | followers |
|---------|----------------------------|----------|-----------|---------|---------|-----------|-----------|
| 0 | '6734b7ec636dd75aa4e16fc2' | 'Nombre' | 'admin' | 'admin' | 'ADMIN' | 0 | 0 |
| 1 | '673dfa424cc36c59934d0205' | 'a' | 'a' | 'a' | 'a' | 0 | 0 |
| 2 | '673e037d06a33cbd1c17d8e0' | 'test' | 'test' | 'test' | 'test' | 0 | 0 |
| 3 | '6742393b6f2f88887d1536ff' | 'admin' | 'admin' | 'adm' | 'admin' | 0 | 0 |
| 4 | '67423a0d6f2f88887d153700' | 'admin' | 'admin' | 'ads' | 'as' | 0 | 0 |
| 5 | '6748f0974d9b08c74ab1fb4d' | 'name' | 'surname' | 'email' | 'nick' | 0 | 0 |
| 6 | '6748f176f61c37e0cd3dcfc1' | 'name' | 'surname' | 'emal' | 'nic' | 0 | 0 |
| 7 | '6748f1d2f3ba2fc4a3291f91' | 'name' | 'surname' | 'ema' | 'ni' | 0 | 0 |

Listar Usuarios (Con query):

```
ADMIN : listUsers -q '{ name : "admin" }'
```

| (index) | id | name | surname | email | nick | following | followers |
|---------|----------------------------|---------|---------|-------|---------|-----------|-----------|
| 0 | '6742393b6f2f88887d1536ff' | 'admin' | 'admin' | 'adm' | 'admin' | 0 | 0 |
| 1 | '67423a0d6f2f88887d153700' | 'admin' | 'admin' | 'ads' | 'as' | 0 | 0 |

Actualizar Usuario (Con 1 único valor)

```
ADMIN : updateUser -n prueba
[LOG] Se ha registrado un cambio en los datos del usuario del usuario: <name>:prueba, <surname>:admin, <email>:admin, <nick>:ADMIN, <password>:admin.
[Éxito] >> Has actualizado tus datos de usuario.
ADMIN : listUsers -q '{ name : "prueba" }'
```

| (index) | id | name | surname | email | nick | following | followers |
|---------|----------------------------|----------|---------|---------|---------|-----------|-----------|
| 0 | '6734b7ec636dd75aa4e16fc2' | 'prueba' | 'admin' | 'admin' | 'ADMIN' | 0 | 0 |

Actualizar Usuario (Múltiples campos)

```
ADMIN : updateUser -n Alejandro -s Pastor -i Alex
[LOG] Se ha registrado un cambio en los datos del usuario del usuario: <name>:Alejandro, <surname>:Pastor, <email>:admin, <nick>:Alex, <password>:admin.
[Éxito] >> Has actualizado tus datos de usuario.
Alex : listUsers -q '{ nick : "Alex" }'
```

| (index) | id | name | surname | email | nick | following | followers |
|---------|----------------------------|-------------|----------|---------|--------|-----------|-----------|
| 0 | '6734b7ec636dd75aa4e16fc2' | 'Alejandro' | 'Pastor' | 'admin' | 'Alex' | 0 | 0 |

```
Alex : █
```

En cuanto a la comprobación de errores también se han analizado todos o casi todos los casos y se han controlado para la salida por consola. De forma que no se rompa el flujo del programa.

Seguir a otro usuario y listar seguidores → Follow & ListFollowing:

```
ADMIN : follow --id 673e037d06a33cbd1c17d8e0
[LOG] Se ha registrado un nuevo follow del usuario con <nick>:ADMIN al usuario con <nick>:test.
[Éxito] >> Has empezado a seguir al usuario con <nick> : test
ADMIN : listFollowing
```

| (index) | id | name | surname | email | nick |
|---------|----------------------------|--------|---------|--------|--------|
| 0 | '673e037d06a33cbd1c17d8e0' | 'test' | 'test' | 'test' | 'test' |

Listar usuario a los que sigues → ListFollowers: (Desde el usuario Test)

```
test : listFollowers
```

| (index) | id | name | surname | email | nick |
|---------|----------------------------|---------|---------|---------|---------|
| 0 | '6734b7ec636dd75aa4e16fc2' | 'admin' | 'admin' | 'admin' | 'ADMIN' |

Dejar de seguir a un usuario → Unfollow: Usuario Admin

```
ADMIN : unfollow --id 673e037d06a33cbd1c17d8e0
[LOG] Se ha eliminado el follow del usuario con <nick>:ADMIN para el usuario con <nick>:test.
[Éxito] >> Dejaste de seguir al usuario con <nick> : test
ADMIN : listFollowing
[Error] >> La búsqueda realizada no ha devuelto ningún resultado. ¿No sigues a nadie? o ¿Búsqueda demasiado intensiva?
```

Usuario Test:

```
test : listFollowers
[Error] >> La búsqueda realizada no ha devuelto ningún resultado. ¿Nadie te sigue? o ¿Búsqueda demasiado intensiva?
```

Tweets:

```
ADMIN : addTweet -c 'Este es el Primer Tweet'
[Éxito] >> Has creado un nuevo Tweet.
ADMIN : [LOG] Se ha registrado un nuevo tweet creado por el usuario con <nick>:ADMIN y con el contenido: Este es el Primer Tweet.
ADMIN : listTweets -q '{content: "Este es el Primer Tweet"}'
```

| (index) | id | owner | content | retweets | like | dislike |
|---------|----------------------------|---------|---------------------------|----------|------|---------|
| 0 | '67491529d7557ac09a8d7229' | 'ADMIN' | 'Este es el Primer Tweet' | 0 | 0 | 0 |

Retweet:

```
test : addRetweet --id 67491529d7557ac09a8d7229
[LOG] Se ha registrado un nuevo retweet enviado por el usuario con <nick>:test del tweet con id: 67491529d7557ac09a8d7229, owner: ADMIN y contenido: Este es el Primer Tweet.
[Éxito] >> Has dado retweet al Tweet publicado por ADMIN
test : listTweets -q '{content: "Este es el Primer Tweet"}'
```

| (index) | id | owner | content | retweets | like | dislike |
|---------|----------------------------|---------|---------------------------|----------|------|---------|
| 0 | '67491529d7557ac09a8d7229' | 'ADMIN' | 'Este es el Primer Tweet' | 1 | 0 | 0 |

Like:

```
test : like --id 67491529d7557ac09a8d7229
[LOG] Se ha registrado un nuevo like del usuario: test al tweet con id: 67491529d7557ac09a8d7229 creado por: ADMIN y con el contenido: Este es el Primer Tweet.
[Éxito] >> Has dado un like al Tweet publicado por ADMIN
test : listTweets -q '{content: "Este es el Primer Tweet"}'
```

| (index) | id | owner | content | retweets | like | dislike |
|---------|----------------------------|---------|---------------------------|----------|------|---------|
| 0 | '67491529d7557ac09a8d7229' | 'ADMIN' | 'Este es el Primer Tweet' | 1 | 1 | 0 |

Dislike: (Eliminación del like previo)

```
test : dislike --id 67491529d7557ac09a8d7229
[LOG] Se ha registrado un nuevo dislike del usuario: test al tweet con id: 67491529d7557ac09a8d7229 creado por: ADMIN y con el contenido: Este es el Primer Tweet.
[Éxito] >> Has dado un dislike al Tweet publicado por ADMIN
test : listTweets -q '{content: "Este es el Primer Tweet"}'
```

| (index) | id | owner | content | retweets | like | dislike |
|---------|----------------------------|---------|---------------------------|----------|------|---------|
| 0 | '67491529d7557ac09a8d7229' | 'ADMIN' | 'Este es el Primer Tweet' | 1 | 0 | 1 |

Con esto todas las funcionalidades han sido logradas.

Para agregar contenido también hemos realizado un Logout:

```
ADMIN : logout
>> ¿Ya te marchas ADMIN? ¡Te veo más tarde!
```

LOGIN MENÚ

1. login -e <email> -p <password>
2. addUser -n <name> -s <surname> -e <email> -p <password> -i <nick>
3. exit

Haz uso de los comandos anteriores para iniciar sesión o registrarse.
Para más información acerca de un comando utiliza <cmd> --help

```
[TW Lite] : █
```

y un Exit:

```
[TW Lite] : exit
>> ¿Tan pronto te vas? Si todavía no accediste a la APP. Bueno... ¡¡Nos vemos pronto!!
```

Lab 1 : Conclusiones

El desarrollo de Twitter Lite ha sido una experiencia enriquecedora que ha permitido explorar de manera práctica conceptos clave en la integración de aplicaciones. Destacar sobre todo la arquitectura en base a módulos implementada que ha facilitado la organización del proyecto sentando las bases para futuras extensiones asegurando que las nuevas funcionalidades puedan añadirse de forma sencilla y sin afectar los componentes existentes.

La utilización de MongoDB como base de datos permite gestionar datos de forma flexible y eficiente optimizando operaciones críticas como la gestión de seguidores y las interacciones con mensajes.

La interfaz de usuario se desarrolló con un enfoque centrado en la experiencia del usuario brindando una navegación intuitiva con menús claros y colores llamativos. Las validaciones detalladas y los mensajes personalizados mejoran la interacción evitando errores comunes y facilitando el uso de la aplicación. Del mismo modo, se prestó especial atención a la robustez del sistema asegurando que los errores fueran manejados de manera controlada para evitar interrupciones en el flujo del programa.

Todas las funcionalidades planteadas al inicio del proyecto se lograron con éxito desde el registro y autenticación hasta la gestión avanzada de tweets de usuarios.

Por último, ha sido una aplicación bastante divertida y educativa.

Lab 2 : Seguimiento del Proceso : Servicio RESTful

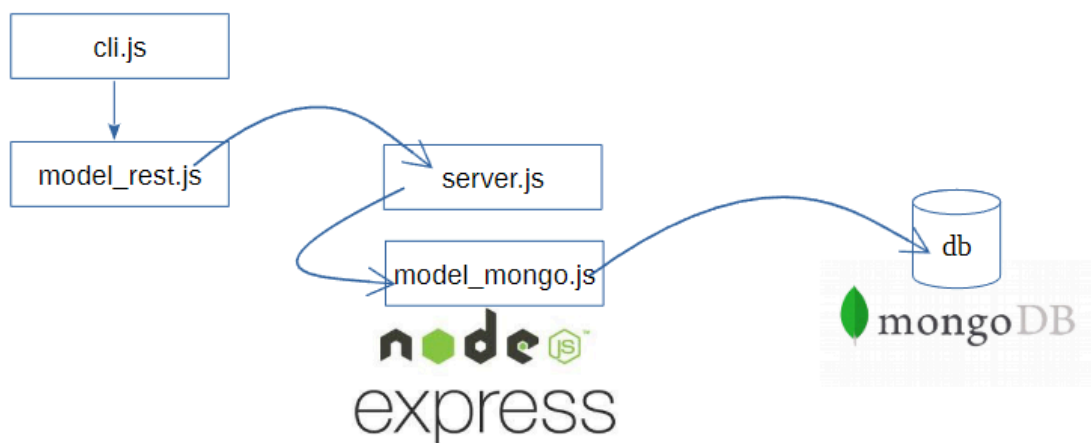
En este segundo laboratorio hemos enfocado como objetivo la creación de un servicio RESTful para una aplicación de **Twitter Lite**, desarrollada previamente con **Node.js** y **MongoDB**, y ahora haremos uso de **Express**. Nuestra aplicación era capaz de proporcionar una serie de operaciones que permitían gestionar usuarios, tweets y sus interacciones (como seguir a otros usuarios, retweets, likes, dislikes), todo ello con un mecanismo de autenticación basado en tokens.

En este segundo laboratorio, nos vamos a centrar en el desarrollo de este servicio mediante dos fases principales: **implementación del servidor RESTful** y **consumo del servicio mediante un cliente**.

El primer paso en la implementación será el diseño de la API, definiendo los **puntos de entrada** y las **operaciones** disponibles. Una vez elaborada pasaremos a la implementación del servicio RESTful mediante Node.js y Express haremos peticiones HTTP con Axios al servidor. Utilizaremos una nueva librería que llamaremos `model_rest` que hará de punto intermedio entre `model_mongo` (almacenamiento) y el CLI (cliente) permitiendo que el cliente pueda realizar acciones en remoto sin necesidad de estar directamente conectado al almacenamiento.

Para probar nuestros métodos podríamos probarlos directamente sobre el navegador pero al requerir Token será muy limitado. Para ello, no es necesario, pero también podríamos utilizar Postman o Thunder para las peticiones HTTP (en caso de ser necesarias que no lo son ya que las haremos desde el cliente con Axios).

Nuestra aplicación deberá tener la siguiente estructura :



Lab 2 : Herramientas

Para el desarrollo de este laboratorio vamos a hacer uso de varias herramientas útiles.

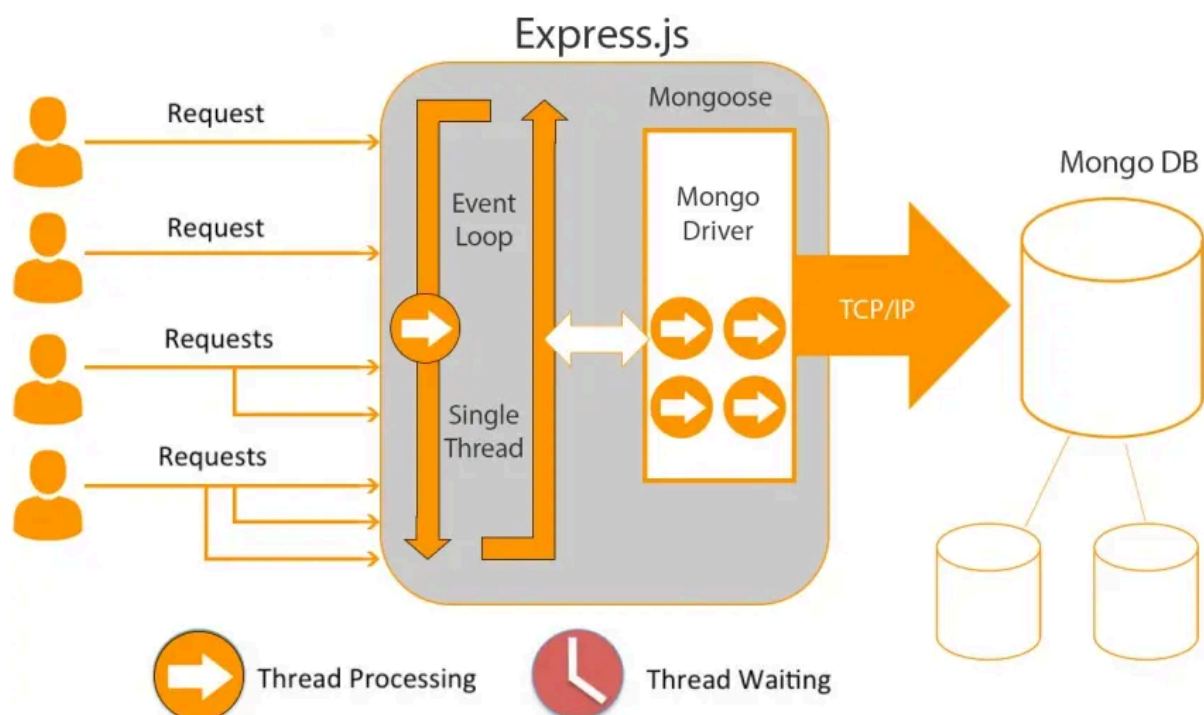
Entre ellas se encuentra **Axios**.

Axios es una librería de JavaScript basada en promesas que se utiliza para hacer solicitudes HTTP desde el navegador o desde Node.js. En nuestro ejemplo, desde Node.



También haremos uso de **Express**, que es un framework minimalista y flexible para aplicaciones web y APIs en Node.js.

Express proporciona un sistema de enrutamiento fácil de usar que permite gestionar rutas y solicitudes HTTP (GET, POST, PUT, DELETE, etc.) de manera eficiente. También permite el uso de *middleware*, que son funciones que se ejecutan en la solicitud antes de llegar a la ruta final. El middleware puede manipular las solicitudes, las respuestas o incluso detener el flujo si es necesario.

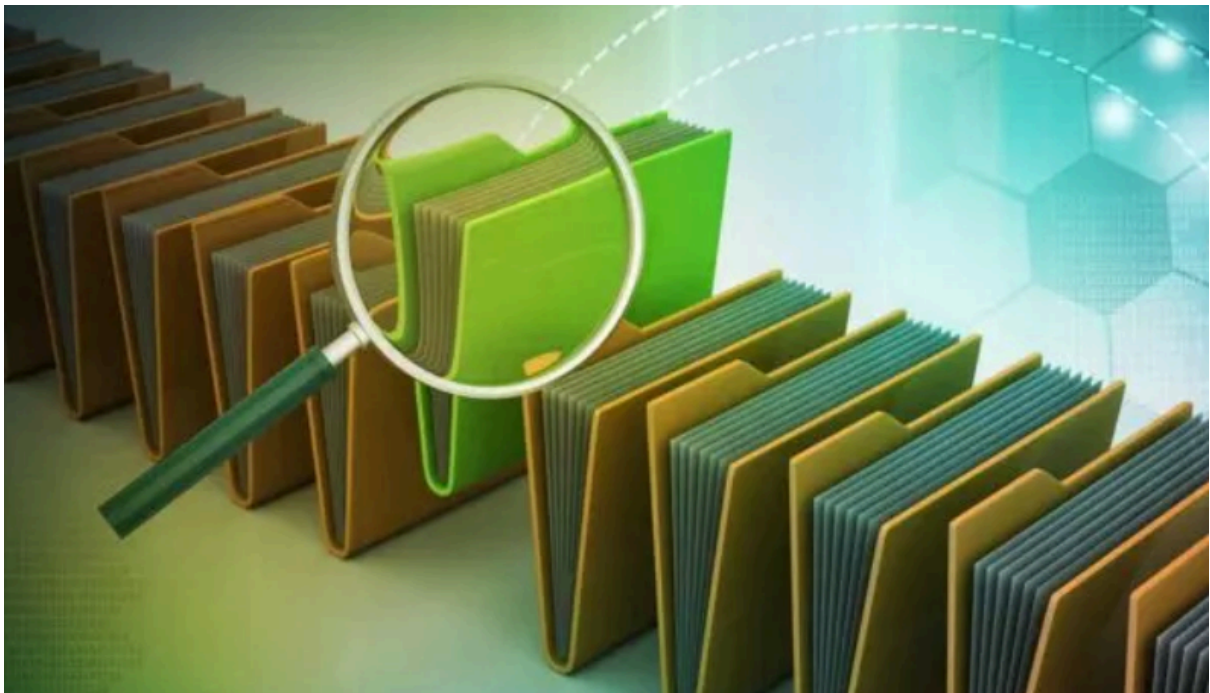


Lab 2 : Nuevas Funcionalidades :

Adicionalmente a lo que ya teníamos diseñado en el laboratorio 1, en este laboratorio 2 hemos agregado variedad de funcionalidades extra aparte de completar el servicio REST.

Sistema de Logs :

Debido a que nuestra aplicación (Twitter Lite) empieza a fomentar gran cantidad de mensajes en el servidor y cliente. Nos hemos visto obligados a implementar un sistema de Logs profesional para la aplicación para registrar todas las acciones que ocurren, por consola y en un fichero.



La utilización de estos logs nos ayudará a determinar posibles errores en algunas funciones y detectarlos con más agilidad y rapidez.

```
258 [12/12/2024 18:15:46] [Server] El usuario con ID <675af516d705ac64960bae06> ha ejecutado el comando listTweets()
259 [12/12/2024 18:20:40] [Server] Servidor iniciado y escuchando en: http://localhost:8080
260 [12/12/2024 18:20:51] [Server] Acaba de acceder a la plataforma el usuario <test> con email: <test>
261 [12/12/2024 18:20:51] [200 Éxito] >> Bienvenido a Twitter test. ¿Qué tal si empezamos a Twittear?
262 [12/12/2024 18:20:57] [Server] El usuario con ID <671cfd7167b81c79cec1bb72> ha ejecutado el comando listUsers()
263 [12/12/2024 18:21:03] logger is not defined
264 [12/12/2024 18:21:34] Problema con la solicitud: El servidor no está iniciado o hay algún error.
265 [12/12/2024 18:21:36] [Server] Servidor iniciado y escuchando en: http://localhost:8080
266 [12/12/2024 18:21:43] [401 Unauthorized] >> El usuario o la contraseña especificados no existen en nuestra base de datos.
267 [12/12/2024 18:22:05] [Server] La aplicación ha registrado a un nuevo usuario: <name>:test, <surname>:test, <email>:test, <nick>:test en la ba
268 [12/12/2024 18:22:05] [201 Created] >> ¡Enhorabuena! te has registrado correctamente en Twitter.
269 [12/12/2024 18:22:07] [Server] Acaba de acceder a la plataforma el usuario <test> con email: <test>
270 [12/12/2024 18:22:07] [200 Éxito] >> Bienvenido a Twitter test. ¿Qué tal si empezamos a Twittear?
271 [12/12/2024 18:22:11] [Server] El usuario con ID <675b1bbdb7a964deba98d4bd> ha ejecutado el comando listUsers()
272 [12/12/2024 18:22:19] [Server] Usuario con <ID>:675b1bbdb7a964deba98d4bd eliminado. Eliminados: Tweets(0), retweets(0), likes(0), dislikes(0),
273 [12/12/2024 18:22:19] [200 Éxito] >> Has eliminado de la base de datos tu propio usuario...
274
```

Métodos :

Del mismo modo, hemos implementado un nuevo método para la eliminación de los usuarios llamada deleteUser.

```
=====
                        MENÚ DE AYUDA : DELETEUSER
=====
• Descripción: Elimina al usuario de la base de datos.
• Uso: deleteUser --id <userID>.
• Variables:
  |--id| : <userID>    -> ID del Usuario al que se quiere eliminar.

• Especificaciones:
  - Si te eliminas el usuario propio se te desconectará de la sesión.
  - Eliminar a un usuario equivale a eliminar toda la información de Tweets / Follows y Likes.

NOTA: La ejecución del comando solamente estará disponible una vez autenticado.

ADMIN : █
```

Eliminar a un usuario conllevará en la eliminación recursiva de todos sus recursos en la aplicación. Es decir, si el usuario dispone de algún tweet, like, follow, etc. Todo esto será eliminado.

Model Rest:

La base de nuestra aplicación ahora se basa en peticiones http a un servidor. El cual estará escuchando por el puerto 8080.

En el código veremos los cambios pero aquí podemos mostrar alguna imagen de la aplicación con la necesidad de tener el servidor abierto:



Errores y Éxitos:

Se ha realizado un cambio en el formato de los errores y las peticiones correctas. Dependiendo de la clase se lanzará un código diferente con un mensaje especial:

```
[TW Lite] : addUser -e prueba -s prueba -n prueba -p prueba -i prueba
[201 Created] >> ¡Enhorabuena! te has registrado correctamente en Twitter.
[TW Lite] : login -e prueba -p prueba
[200 Éxito] >> Bienvenido a Twitter prueba. ¿Qué tal si empezamos a Twittear?
```

Del mismo modo, para los errores:

```
[TW Lite] : addUser -e prueba -s prueba -n prueba -p prueba -i prueba
[409 Conflict] >> No se ha podido completar el registro debido a que ya existe un usuario con ese email registrado.
[TW Lite] : addUser -e prueba -s prueba -n prueba -p prueba
[400 Bad Request] >> No se ha detectado el parámetro -i <nick>. Haz uso de addUser --help para más información.
[TW Lite] : login -e test -p te
[401 Unauthorized] >> El usuario o la contraseña especificados no existen en nuestra base de datos.
[TW Lite] :
```

También para los informativos:

```
ADMIN : listUsers -q '{name : "asd"}'
[100 Info] >> No existen resultados con esos criterios de búsqueda.
ADMIN : listFollowers
[100 Info] >> La búsqueda realizada no ha devuelto ningún resultado. ¿No sigues a nadie? o ¿Búsqueda demasiado intensiva?
ADMIN :
```

Todos estos mensajes también serán almacenados en el archivo de Log para un correcto funcionamiento de la aplicación.

Lab 2 : Conclusiones

Creo que ha sido un laboratorio entretenido y sobre todo esencial para aprender a realizar una api con un servicio Restful.

Pese a que el proceso ha sido sencillo, debido a que ya disponíamos de una base (el laboratorio 1) y simplemente había que extenderla y realizar pequeños cambios, creo que ha habido momentos en los que algunas cosas se medio complicaba, como estar atento a los parámetros de Axios en la URL o en el Body. Aunque es de agradecer que la lógica de negocio / almacenamiento apenas la hemos modificado / adaptado.

Desde mi perspectiva, dónde parto desde no haber realizado nunca un EndPoint, no saber que es un Middleware o apenas saber qué es Body-Parser, puedo decir que me ha servido de mucho este laboratorio y sin duda muy esencial para mi futuro y aprendizaje.

Dejamos el código bien preparado y organizado para el próximo laboratorio. Dónde implementaremos colas de mensajes.