Minitweeter

Introducción

Como objetivo de resolver el challenge técnico solicitado por Ualá, se desarrolló en una aplicación web una mini plataforma de microblogging similar a twitter. Dicha aplicación se encuentra subida al repositorio de github https://github.com/gab-rod23/minitweeter, para que pueda ser revisada y evaluada.

Requerimientos

Los requerimientos solicitados y desarrollados para que la aplicación satisfaga la entrega son los siguientes:

- Que un usuario pueda publicar mensajes cortos (tweets) que no excedan un límite de 280 caracteres.
- Que un usuario pueda seguir a otro usuario.
- Que un usuario pueda ver de forma ordenada una linea de tiempo de tweets que hayan publicado los usuarios a los que sigue.

Con el fin de facilitar las pruebas, se agregaron las siguientes funcionalidades extras:

- Poder dar de alta a un usuario.
- Poder consultar la información de un usuario.

Especificaciones

La aplicación **minitweeter** se encuentra desarrollada en lenguaje **Golang** versión 1.21.13. Se decidió utilizar está tecnología gracias a sus cualidades de ser un lenguaje ideal para construir **sistemas escalables** y con **baja latencia**.

La aplicación se construyó utilizando el framework **Gin Gonic**, utilizado para el desarrollo del servidor, y el servicio de almacenamiento de **MongoDB** para persistir los datos tanto de los usuarios como de los tweets publicados.

La estructura de los archivos fuentes de la app se encuentran distribuidos dependiendo del dominio del negocio al que pertenezcan, haciendo uso del concepto de **Clean Architecture**. En cada dominio, los archivos se agrupan en distintos niveles de abstracción:

- ***Controller:*** donde se encuentran funciones "handler" de cada endpoint y se realiza la validación de los request entrantes.
- ***Usecase:*** donde se encuentra toda la lógica de negocio.
- ***Repository:*** donde se especifican todas las interacciones que tiene la aplicación con la base de datos.
- ***Entities:*** se almacenan las entidades como modelos y dtos.

Por último, como información extra, se detallan los frameworks y dependencias utilizados para la construcción de la aplicación:

- Gin gonic: para montar el servidor web y que reciba peticiones http.
- Mongo-go-driver: para realizar la conexión e interactuar con la base de datos construida en mongodb.
- Devfeel/mapper: para agilizar el mapeo de atributos entre estructuras distintas.

Funcionalidades

➔ Creación de usuarios:

Se realiza mediante una petición POST, enviando la siguiente información:

- Usuario
- Nombre
- Correo electrónico

```
{
  "username": "jorgelin",
  "name": "Jorge Perez",
  "mail": "jorge@hotmail.com"
}
```

Una vez recibida la petición el servidor valida de que tanto el usuario como el correo electrónico no se encuentren utilizados por un usuario ya existente. Si ese no fuera el caso el servidor inserta el registro del usuario en la base de datos y devuelve una respuesta de creación exitosa (201). En caso contrario, devuelve una respuesta informando que el usuario/correo electrónico ya se encuentran en uso.

➔ Obtener datos del usuario:

Se realiza mediante una petición GET, enviando el siguiente header:

- username: <USUARIO_A_CONSULTAR>

Una vez recibida la petición el servidor realiza la consulta a la base de datos solicitando la información del usuario. En caso de que exista, devuelve los siguientes campos:

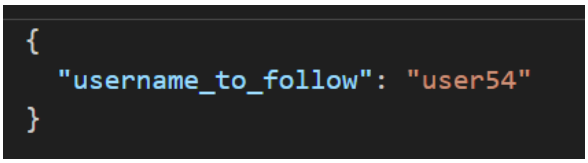
- Usuario
- Nombre
- Correo electrónico
- Fecha de creación
- Seguidores (followers)
- Usuarios a los que sigue (following)

En caso de que no exista el usuario a consulta, el servidor responde con un código 404 (Not found) y un mensaje informando que el usuario no existe.

➔ Seguir usuario:

Se realiza mediante una petición POST, enviando la siguiente información en body:

- Usuario a seguir



Y enviando el siguiente header:

- username: <USUARIO_QUE_SOLICITA_SEGUIR>

Al recibir la petición, el servidor genera un **bloqueo** de consulta del *usuario a seguir* y del *usuario que solicita seguir* en la base de datos, para evitar que durante todo el proceso que conlleva el agregado de un seguidor otra conexión intente recuperar información de algunos de los usuarios involucrados, la cual estaría desactualizada.

Luego del bloqueo el servidor valida que ambos usuarios existan, y recupera la lista de usuarios siguiendo (*following*) del *usuario que solicita seguir* (username) y la lista de seguidores (*followers*) del *usuario a seguir* (username_to_follow). Un vez obtenida estas listas se realiza la siguiente operación:

- Se agrega a **username** a la lista de *seguidores* de **username_to_follow**
- Se agrega a **username_to_follow** a la lista de *siguiendo* de **username**

Una vez realizado esto, se actualizan los registros correspondientes a cada usuario en la base de datos y se **desbloquean** los mismos.

➔ Crear tweet:

Se realiza mediante una petición POST, enviando la siguiente información en body:

- Texto del tweet

```
{
  "text": "este es mi tweet"
}
```

Y enviando el siguiente header:

- username: <USUARIO_QUE_REALIZA_TWEET>

Una vez recibida la petición el servidor primero valida que el texto del tweet no sea mayor a 280 y luego que el usuario exista en la base de datos. Una vez pasada las validaciones se almacena el tweet en la base de datos con los siguientes atributos:

- Usuario
- Texto del tweet
- Fecha y hora de creación del tweet

```
_id: ObjectId('66ba9ba5aec523c8b973f325')
username : "gabirod92"
text : "este es mi quinto tweet!"
created_date : 2024-08-12T23:32:53.727+00:00
```

➔ Obtener timeline:

Se realiza mediante una petición GET, enviando los siguientes headers:

- username: <USUARIO_A_CONSULTAR>
- page_size: <TAMAÑO_DE_PAGINA_MAYOR_A_0>
- page_number: <NUMERO_DE_PAGINA> (opcional, por defecto 0)
- last_page_tweet_date: <FECHA_ULTIMO_TWEET_DE_PAGINA> (opcional)

username	pepito
page_size	5
page_number	0
last_page_tweet_date	2024-08-12T23:32:58.091Z

Una vez recibida la petición el servidor valida que el usuario exista y recupera la lista de usuarios a los que sigue. A partir de esa lista recupera una cierta cantidad de tweets

pertenecientes a cada uno de esos usuarios en formato de ***lista ordenada, de manera descendente, por fecha de publicación***, y los devuelve como respuesta.


```
{
  "Timeline": [
    {
      "text": "este es mi quinto tweet!",
      "username": "gabirod92",
      "created_date": "2024-08-12T23:32:53.727Z"
    },
    {
      "text": "este es mi cuarto tweet!",
      "username": "gabirod92",
      "created_date": "2024-08-12T23:32:46.142Z"
    },
    {
      "text": "este es mi tercer tweet!",
      "username": "gabirod92",
      "created_date": "2024-08-12T19:52:57.412Z"
    },
    {
      "text": "este es mi primer tweet!",
      "username": "gabirod92",
      "created_date": "2024-08-12T19:26:53.216Z"
    }
  ]
}
```


La cantidad de tweets que devuelve la consulta está determinada por los headers **page_size** y **page_number**, ya que se realiza un paginado a partir de estos. El valor **page_size** indica cuántos tweets se van a devolver por cada consulta y el valor **page_number** indica qué página se va a devolver. Para obtener la página deseada se saltean registros a partir del cálculo ***page_size*page_number***. El header **last_page_tweet_date** se utiliza como referencia para poder navegar a la siguiente

página, ya que los tweets que se encuentren en la misma van a tener una fecha de creación menor a **last_page_tweet_date**.