Twitter exercise

Nicolò Pomini

14 febbraio 2018

Indice

1 Introduction

This document provides a simple description of a *Twitter like* `API`, for an internship interview with `U-Hopper`.

2 Content

2.1 Assumptions

For simplicity, I made some assumptions.

- Firstly, a tweet is composed only by plain text, so no special character - like *hastags* or *tags* - are recognized.
- Secondly, no user handling is required. By assumption users are already defined and stored somewhere, and from APIs is not possible to add or modify users.
- Finally, every tweet is public, therefore APIs can be used by everyone, with no access-control.

2.2 Architecture

The application is structured following the MVC pattern.

Model defines data schemas, all the essential to handle users and tweets.

Control offers the accessable routes and handles requests.

View this part is done just by JSON objects, because this application is only server-side and is not required a human-friendly interface.

The application makes use of an external database, for data persistance.
The application offers a `REST` service, totaly uncoupled and uniterested with client.

3 Authentication

If the environment would require authentication every API should have a specific parameter to recognize the author of the request. In my opinion, there are two methods to authenticate users:

- The stronger method is using a token. A token is an alpha-numeric string, arbitrarily long, unique among all users. Before perform a request, a user has to log into the system and requires a token. For an higher level of security, tokens can also have a time-expiration, so before every request user has to request a new token. This method is quite onerous to implement, but offers a two-level authentication.
- An easier way is to use the user ID to identify every request. This method is not applicable if is used a simple *Integer auto-increment* ID, because with a bit of brute force is easy to find a valid ID. But if is used an alpha-numeric ID, like *Mongoose* `ObjectId` it should works. In fact, if the ID is generated with an alphabet of x symbols, the length of an ID is n and there are u valid users, the probability to guess a valid ID is

  $$\frac{1}{x^n}u$$

  which with an alphabet of 31 symbols and not too many users is enough small.

4 Conclusions

For the implementation I choose to use `Node.JS` because I feel confident to use this language to implement a `REST` service, in couple with `Mongoose` for `MongoDb`, for data storage.