Robbot project documentation

Roberto Antoniello

April 4, 2023

In this file I resume how Robbot works with more details than the README file you can find in the repository. **Robbot** is a Telegram bot with a bunch of different features. Some of them extend the use of Telegram you usually do, some of them are just funny features I liked to develop.
The bot is written in Python and it is running in Python3.9 version at the moment.

1 Configuration file

The configuration file is the first thing you must check because without it, the bot won't run at all. This file is also well explained in README file in this repository so I won't spend a lot of words on it.
Essentially you put your keys, additional data and commands' name in the right fields to make the bot work.

$$config - example.json \Rightarrow config.json$$

1.1 DB path

The .db file will be saved in the directory you choose and set in config.json file

$$"path - db" \Rightarrow config.json$$

1.2 Super admin data

In this section we put our Telegram data as super admin(the only one) of the bot. The super admin is able to manage the db changes such as adding users, promote to admin and deleting users/revoke admin powers.
This type of user can also modify the value of a few data like the number of uses of commands for a specific user or managing the amount of uses for openai commands of premium users.

$$"id - super - admin" \Rightarrow config.json$$

1.3 commands,admin commands and super

Here we put the name of commands the bot is able to recognize and execute. For example if we don't want to execute the weather we just don't put it on user-commands field. It's the same for admin and super commands.

$$"user - commands" \Rightarrow config.json$$
$$"admin - commands" \Rightarrow config.json$$
$$"super - admin - commands" \Rightarrow config.json$$

2 Main

app.py is the main source file. It creates the session, it connects with Telegram API using Pyrogram Client class and it's in constant wait for arriving messages. First of all it fetches the information from configuration file, then it starts the connection.
While waiting, the bot will ignore any messages it doesn't recognize as a known command. Otherwise if it's a known command, it checks for permission to the user launched that command. If it's a recognized user, the entire message will be managed by a parser function and moved to a specific fetch function written inside **controller.py**.
If a command isn't launched by a registered user, the bot will reply with a default message.

$$app.py \xrightarrow{\text{command}} controller.py$$

3 Controller

Just like in the MVC design pattern, this controller takes the input arriving from the main and it calls the right function in the right source file in *utils* and *modules* folders. How it works?
There are three dictionaries dedicated to the three types of commands which are as said in before **(user,admin,super)**.
The key is the command and the value is the path of function it executes.
As said in Main section, here we have the fetch command functions matched with the dictionaries mentioned in the previous row. The specific function will return a **sendMessage** or a **sendMedia** function and then the main will wait for the next command.

$$controller.py \xrightarrow{\text{command fetched}} (modules\ or\ utils)$$

3.1 parser function

This simple function mentioned in the previous sections takes the input message as argument and it deletes the command at the beginning, returning then just the text "query" that will be sended by the controller to the right module which will manage and execute it.

3.2 other details

Inside the controller there is also a small function called **visualizza** which print the current state of the incoming message on terminal such as a live log.
During the execution of this function, the log is also sended as a Telegram message to the super admin private chat, including possible errors/exceptions.

4 get-config

this source file contains some support functions that are often used in many situations. The first one return in a variable the content of the configuration file to get the fields easily.
The remaining functions are all about Telegram features such as getting information from the json message or renaming for example the *send-message* of **Pyrogram** to avoid including the decorator in every source file and just call this one by including get-config.

$$app.py \xrightarrow{\text{get-config-file()}} running$$

5 sysfunctions

This file contains functions using directly Telegram features such as the poll function or get-message. It also includes the help function.

6 Summary till now

$$app.py \xrightarrow{\textbf{get-config-file()}} running$$

$$app.py \xrightarrow{\textbf{command}} controller.py$$

$$controller.py \xrightarrow{\textbf{command fetched}} (modules\ or\ utils)$$

$$(modules\ or\ utils) \xrightarrow{\textbf{return message or media}} app.py$$

$$app.py \xrightarrow{\textbf{waiting for the next command}} app.py$$

7 Database

In this section we'll focus about the database structure. There are several tables linked each other: **User, Stats, Trivial, Group, OpenAICredit**.

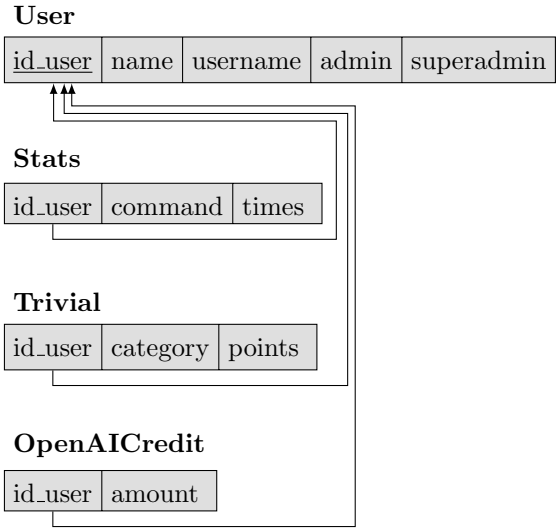
The **User** table saves data of users(Telegram id, first name and username(if it exists otherwise None value), there are also a couple of boolean field to set admin or superadmin rights.

The **Stats** table is linked to the User one by the id field and it saves information about how many times a command is used by every user.

The **Group** table saves chat-id, chat-name and a third string field. This table is used to save a specific command to execute only in specific group chats.

The **Trivial** table is used to manage the score points of the Trivial game developed inside this bot. It's linked to the User table by the id as foreign key.

The **OpenAICredit** table is linked to the User table by the id as foreign key, just used to manage number of uses of premium commands for premium users.



The **Group** table isn't linked to any table so it isn't shown in this figure. There's also an additional table called **TrivialSavedData** but it is standalone like the Group one. It is used to save the current active trivial quiz.