This file contains key information over the bot and the database.

Quick Disclaimer

The bot is going to need admin-level privileges as it will be creating private threads for the user to send their private/public API key. This is the only way to keep all information confidential to the user. The channel in which the thread is created **must** be a public text channel. Private threads, by default, are allowed to be seen by mods and admins.

Furthermore, the bot has been written to accept both standard and unified account types. The user, when giving the bot their private/public key will also have to specify whether or not they hold a standard account or a unified account.

Project structure

The files includes with the project are:

```
bot.py
bot_variables.py
by_bit.py
database.py
database_connection.yaml
table_layout.yaml
dt.yaml
```

`bot.py`

This file is the core of the bot. This is where all commands are declared and where all core functionality for each command is implied.

The file includes a lot of fancy Python workings, such as wrappers, to deter from repetitive code. You will find names such as (names span across multiple files):

- `BV` - This stands for Bot Variables. This is the abbreviation given to the `BotVariables` class found in `bot_variables.py`.
- `HTBP` - This stands for Has To Be Private. This is passed to `check_permissions` function, found in the `BotVariables` class in `bot_variables.py`. If this is `True`, the command *has* to be invoked in a private channel; inversely, if it is `False` the command *does not* have to be invoked in a private channel, but admin-permissions still need to be held by the member of whom is invoking the command.
- `WPTE` - This stands for With Private Thread Exception. This is passed to `public_channel_required` function, found in the `BotVariables` class in `bot_variable.py`. If this is `True`, the command *has* to be invoked in a private thread; inversely, if this is `False` the command *does not* and *can not* be invoked in a private thread.
- `BE` - This stands for Bot Event (`bot.event` as it is known with discord.py). This, too, is found in the `BotVariables` class in `bot_variables.py`. This is

just a way to shorten things up. It is not needed, but I implemented it anyway - it got tedious writing out `bot.event` over and over.
- `db` - This stands for Database. This is found in the `BotVariables` class in `bot_variables.py`. This variable is the key point to all database interaction - from adding a user, getting user data, deleting a user etc.

`bot_variables.py`

This file stores all key variables, functions and wrappers for the bot. Instead of cluttering `bot.py` with all the variables, functions and wrappers they are placed in a class so they have a overarching name that makes it easier to identify the variables purpose (which should be implied, but makes it easier to write the code).

`by_bit.py`

This file is the absolute backend for all things in relation to the ByBit API. When the user runs `!submit` followed by the type of account they hold, their private/public API keys the bot will initialize a session for them in an array with the name `sessions`.

This file includes all functionality needed for the bot to perform as requested.

`database.py`

This file is the absolute backend for all things in relation to the database. Functions such as `add_user`, `remove_user` etc can be found in this file.

`database_connection.yaml`

This file has been implemented to make it easier on behalf of the receiver to connect the database to the bot without having to edit any sort of code.

The file has three values:

1. `host` - Or better known as the database instance, this is the host (instance name) of your database. I assume you have some fundamental knowledge over what this implies.
2. `user` - The username linked to the database instance (normally root).
3. `pswd` - The password to the username.

`table_layout.yaml`

This file is where you can specify the table layout you want for your database.

- `database_name` is your preferred name for the database where the table will be stored.
- `table_name` is your preferred name for the table that will store all of the columns (username, public key, private key & account type)
- `columns` is where you can describe all columns in the table.
  - Format is as follows:
    - `column_name : "datatype"`
      - Example: `user: "varchar(15)"` will create a column titled user with a datatype varchar that allows up to 15 characters.

`dt.yaml`

This file is where you can put the discord token that is needed for the bot to run in the server.

Setting Up The Bot

The bot comes equipped with 4 commands that can be used anywhere. The rest of the commands are off limits to all channels by default. The 4 commands that can be used anywhere are "configuration" commands.

Configure the bot with public channels

To add public channels to the bot, invoke the `!add_pub_channel` command. This command requires a channel ID as an argument; the full command is `!add_pub_channel [channel_id]`. Only add public channels where you will allow the `!explain` command, `!ready` command and `!submit` command to be used. Inversely, if you want to remove a public channel run the `!rem_pub_channel` command. This command, too, requires a channel ID as an argument; the full command is `!rem_pub_channel [channel_id]`.

Configure the bot with private channels

To add private channels to the bot, invoke the `!add_priv_channel` command. This command requires a channel ID as an argument; the full command is `!add_priv_channel [channel_id]`. **Make sure** the channel ID passed to this command is, in fact, a private channel. Inversely, if you want to remove a private channel run the `!rem_piv_channel` command. This command, too, requires a channel ID as an argument; the full command is `!rem_priv_channel [channel_id]`.

Once you have your channels configured for the bot, you can invoke the admin-related commands in the private channels in which you configured and the `!explain` command can be invoked in any of the channels configured (private and public). Client-oriented commands such as the `!ready` command and the `!submit` commands can be invoked in the public channels configured; `!submit` is restricted to being invoked only in threads nonetheless. It is advised to invoke the `!explain` command in a public channel in which you are okay with threads being opened in.

Client-oriented commands

As requested, there are only a select few commands geared towards users that will enable them to "register" themselves in the server. These commands can *only be ran once* by the user. With that said, however, admins have commands that can allow users to re-run the commands if need-be.

`!ready` command

In the channel in which the description on how to get a Read-Only API is displayed is where this command is to be invoked. If the channel where `!explain` has been invoked is the only public channel configured with the bot, the `!ready` command will only work in that channel.

The `!ready` command tells the bot the user has followed all the steps and has their private/public keys ready. This command will prompt the bot to create a private thread with the users username as the name. In this thread the bot will ping the user and describe how they can submit their information.

After the user invokes the `!ready` command, their ID will be added to an array called `READY_COMMAND_USED_BY`. Their ID will live in this array unless a admin explicitly removes it. So long as their ID exists in this array, they cannot re-run the `!ready` command.

If a user runs into issues, an admin (and/or mod, depending on permissions) can run `!remove_user`. This command requires two arguments:

- `[from]` - This value can be either `database` or `RCUB` (`READY_COMMAND_USED_BY`).
  - If you run `!remove_user database [user_id]`, the user will be removed from the database. So **be cautious** with this command.
  - If you run `!remove_user RCUB [user_id]`, the user will be removed from the array `READY_COMMAND_USED_BY` thus allowing the user to re-run the command.
- `[user_id]` - The users ID.

`!submit` command

When the user is ready, they will run the `!submit` command. Upon the command being invoked, the user will temporarily be added to the array called `SUBMIT_COMMAND_USED_BY`. This takes place so the user can't re-run the command in the midst of the bot registering them/after the bot registers them. The thread stays alive for a minute after registration is done. At the end of the minute, the thread is deleted and the user is removed from the array. Since they are not allowed to reuse `!ready`, and since `!submit` is only able to be used in private threads, the user will not be able to use either commands again.

The `!submit` commands requires the following arguments:

- `[account_type]` - This is the users ByBit accounts type.
  - For unified accounts:
    - `UC` - CONTRACT
    - `UU` - UNIFIED
    - `UF` - FUND
  - For classic (standard) accounts:
    - `SS` - SPOT
    - `SC` - CONTRACT
    - `SO` - OPTION
    - `SF` - FUND
- `[public_key]` - The users public API key.
- `[private_key]` - The users private API key.

It is of utmost importance to state that the order in which the commands are explained are the order in which they must be passed. `!submit [account_type] [public_key] [private_key]`.

Once the `!submit` command is ran, the users ByBit session (HTTP session) is appended to an array called `sessions` (found in `by_bit.py`). The array stores JSON objects (Python dictionaries). The format is as follows:

```
[
    {
        "user_id": [users_id] (INTEGER),
        "session": HTTP(
            testnet=false,
            api_key=[users_public_api_key],
            api_secret=[users_private_api_key]
        )
    }
]
```

For more information on what `HTTP` is, see https://github.com/bybit-exchange/pybit/blob/master/README.md

Logs

If you want the bot to log all user registration, after configuring the public/private channels via `!add_pub_channel` and `!add_priv_channel` you can run `!enable_logs`. After running `!enable_logs` you will have to tell the bot what channel you want the logs to go into. You can do this via `!set_log_channel [channel_id]`. It is advised that the channel ID in which you pass to `!set_log_channel` is referencing a private channel.

To disable logs, simply run `!disable_logs`. If you want to re-enable logs, you don't have to re-run `!set_log_channel`. That setting will be constant once set.

Testnet

If you, for some reason, want to use testnet the bot comes with a command that allows you to use it.

Simply run `!use_testnet` if you want to use it. If you run this command and later decide you do not want to use it, simply run `!disable_testnet`.

Calculations

As requested, the bot comes equipped with the command `!calc`. This command is a bit complex in regards to its commands.

This command accepts the following commands:

- `[based_on]` - This argument tells the bot what data to base the calculation off of.
  - `bs` - This stands for buy/sell. This tells the bot to base the calculation purely off of the users market buy/market sell orders. It will subtract all the market buys from the users initial balance and, if there are any sells, it will then add the sells to the user account.
  - `s` - This stands for stocks. This tells the bot to base the calculation based off of sell data/stock data over a specified interval (default is

a day). With this, the bot will add together all the market sell orders and then gather all the symbols in which the user is actively investing in. It will then get the market data per each symbol and perform calculations. It will base calculations off of open price/closing price.

- `[user_id]` - The users ID which you want to perform calculations on.
- (OPTIONAL) `[interval]` - If `[based_on]` is `bs`, this command is not needed. If `[based_on]` is `s`, this command, again, is not needed unless you want to explicitly state the interval. By default, the interval is set to `D` (day) meaning the bot will gather market data that is in accordance to the market throughout the day.
  - Upon `[based_on]` being `s` and wanting a different interval, visit https://bybit-exchange.github.io/docs/v5/enum#interval to get a in-depth idea on what intervals are accepted.

Get a overview of users API

If you want a JSON formatted look at a users API key information, run `!get_user_api_info`; this command requires the users ID. The full command is `!get_user_api_info [user_id]`.

An example response will be:


```
{
    "retCode": 0,
    "retMsg": "",
    "result": {
        "id": "13770661",
        "note": "readwrite api key",
        "apiKey": "XXXXXX",
        "readOnly": 0,
        "secret": "",
        "permissions": {
            "ContractTrade": [
                "Order",
                "Position"
            ],
            "Spot": [
                "SpotTrade"
            ],
            "Wallet": [
                "AccountTransfer",
                "SubMemberTransfer"
            ],
            "Options": [
                "OptionsTrade"
            ],
            "Derivatives": [],
            "CopyTrading": [],
            "BlockTrade": [],
            "Exchange": [],
            "NFT": [],
            "Affiliate": []
```

```
        },
        "ips": [
            "*"
        ],
        "type": 1,
        "deadlineDay": 66,
        "expiredAt": "2023-12-22T07:20:25Z",
        "createdAt": "2022-10-16T02:24:40Z",
        "unified": 0,
        "uta": 0,
        "userID": 24617703,
        "inviterID": 0,
        "vipLevel": "No VIP",
        "mktMakerLevel": "0",
        "affiliateID": 0,
        "rsaPublicKey": "",
        "isMaster": true,
        "parentUid": "0",
        "kycLevel": "LEVEL_DEFAULT",
        "kycRegion": ""
    },
    "retExtInfo": {},
    "time": 1697525990798
}
```


If an error occurrs, it will be on behalf of invalid API private/public keys, an internal server error on behalf of ByBit or a wrong user ID. You will be prompted with an error message that states an error has occurred and what the error could derive from.

Show database data in JSON format

If you would like to see the database information in a JSON format run `!show_db`. This command requires no additional arguments. The response will look like:

```
[
    {
        "user_id": [users_id],
        "account_type": [account_type],
        "private_key": [private_key],
        "public_key": [public_key],
        "wallet_balance": [wallet_balance]
    }
]
```

Manually update the users wallet balance

If you want to make sure the databse is up to date in regards to the users wallet balance, run `!update_wallet_balance`; this command requires the users ID. The full command is `!update_wallet_balance [user_id]`.

Questions or Concerns

If you have questions, concerns or ideas on how to enhance the bot do not hesitate to contact firekuramax or email me directly @ mocacdeveloper@gmail.com

ByBit IP bans any IP address from the U.S., in which I reside - so testing has been limited on my behalf.

All work on behalf of fixing bugs will not be subject to extra costs; all work on behalf of new features will be subject to extra costs.