DAW-GPT PYTHON CODE OVERVIEW

- `RefreshAvailableBalance()`: This function is performed through a Flask web application when the URL "/RefreshBalances" is invoked. The function starts by retrieving interest rates from a document in the database called "Info_collection". Next, an empty dictionary called "NewAvailableBalnances" is created to store users' new available balances.

  After that, a loop is performed on all users in the "users_collection" collection. During the cycle, the following calculations are made:

  - The user's return on personal investment is calculated by multiplying the user's current balance by the monthly interest rate for investments.
  - Returns on investment of users of the current user's network are calculated. This is done using an external function called "BuildNetworkBalances" and network layer balances are assigned to the variables `Balance1, Balance2, Balance3,` and `Balance4`. Next, the returns for each network tier are calculated by multiplying the balances by the respective monthly interest rates for the network tiers.
  - Total network profits are calculated by adding up returns for all network levels.
  - Total profits are calculated by adding the return on personal investment and total network profits.

  Finally, the total balance calculated for each user is entered into the "NewAvailableBalnances" dictionary. The user name and calculated values for debugging purposes are also printed.
  Note: The last lines of code beginning with "# for user_id, increment in NewAvailableBalnances.items():" and "users_collection.update_one(...)" are currently commented out. These lines could be used to update users' available balances in the database, but they are currently disabled in the code provided.

- **User `Class`:** This class represents a user and is used to authenticate the user through Flask-Login. The user object has three attributes: `id` (username), password (user `password`), and `AffiliateCode` (user affiliate code).

- **`load_user(user_id)` function:** This is a callback function used by Flask-Login to load the user object based on the user ID. Takes the user ID as an argument and returns a corresponding `User` object if the user is found in the database, otherwise it returns None.

- **`register()`** function: This function is performed through a Flask web application when the URL "/register" is invoked with the GET or POST methods. If the method is POST, the function obtains the username, password, and InvitedBy from the JSON request. The following steps are then done:
  - It is checked if the username is already present in the user database. If it is, a JSON error message is returned.
  - It checks whether the InvitedBy (reference code) is valid. If it is not, a JSON error message is returned.
  - A new affiliate code is generated by calling the **`GenerateAffiliateCode()`** function. It verifies that the generated code is not already present in the list of affiliate codes in the database, and if it is, a new code is generated until a unique code is found.
  - The list of affiliate codes in the database is updated by adding the new generated code.

    - A user document is created with all the necessary information (username, password, current balance, available balance, public key, transactions, affiliate code and reference code) and is placed in the "users_collection" collection in the database.

  Finally, a JSON confirmation message is returned to indicate that the registration was completed successfully.

- **login():** This function is performed through a Flask web application when the URL "/login" is called with the GET or POST methods. If the method is POST, the function obtains the username and password from the JSON request and searches the database for a matching user. If the user is found, a corresponding `User` object is created and the user is authenticated using the `login_user()`. Next, the user is redirected to the "/dashboard" page. If the user is not found or the credentials are invalid, a JSON error message is returned.

- **Logout()** function: This function is performed through a Flask web application when the URL "/logout" is invoked. The function logs out the current user using the **`logout_user()`** function of Flask-Login and redirects the user to the login page.

- **Dashboard()** function: This function is performed via a Flask web application when the URL "/dashboard" is invoked. The function retrieves the current user object using **`current_user`** of Flask-Login. Next, the user document is retrieved from the database based on the user ID. If the user document is found, the desired values such as current balance, available balance, public key, and transaction list are extracted. Finally, the values are returned as JSON or displayed in the "dashboard.html" template for displaying the user's dashboard.

- **update_public_key ()**: This function is performed through a Flask web application when a POST request is sent to the URL "/update_public_key". The function obtains the public key from the request form input and updates the current user's public key in the database. After that, the user is redirected to the "/dashboard" page.

- **Reinvest_balance ():** This function is performed through a Flask web application when a POST request is sent to the URL "/Reinvest_balance". The function gets the amount to reinvest from the request form input, checks whether the amount is less than or equal to the current user's available balance, and calculates the new available balance and the new current balance based on the amount to be reinvested. The values are then updated in the database for the current user, and the user is redirected to the "/dashboard" page.

- **WithdrawCrypto()** function: This function `is executed through a Flask web application when a POST request is sent to the URL "/WithdrawCrypto"`. The function gets the amount to withdraw from the request form input, checks whether the amount is less than or equal to the current user's available balance, and obtains the recipient's address (public key) from the database. If the amount is valid, the `WithdrawBalance()` function is called to process the cryptocurrency withdrawal. Finally, the user is redirected to the "/dashboard" page.

- **Stacking()** function: This function is performed through a Flask web application when the URL "/Stacking" is invoked. The function retrieves the current user object using Flask-Login current_user and obtains a list of stacking packages from the "Stacking_collection" collection in the database. Finally, the function returns the "Stacking.html" template with the list of stacking packages and the wallet key (WALLET_KEY).

- **Affiliates()** `function` : This function is performed through a Flask web application when the URL "/Affiliates" is called. The function retrieves the current user object using `current_user` of Flask-Login and calls the **`build_invites_chain()`** `function` to get the invitation chains for the current user's levels 1, 2, 3, and 4. Finally, the function returns the "Affiliates.html" template with the invitation chains for the various levels.

- **`Rewards()`** function: This function is performed via a Flask web application when the URL "/Rewards" is invoked. The function retrieves the current user object using Flask-Login current_user and obtains budget balances for network tiers 1, 2, 3, and 4 by calling the **`BuildNetworkBalances() function`**. After that, calculate the total earnings on the network and personal earnings based on the balance sheet balances obtained. Finally, the function returns the "Rewards.html" template with total earnings on the network and personal earnings.

- **`Transactions()`** function: This function is executed through a Flask web application when the URL "/Transactions" is invoked. The function retrieves the current user object using **`current_user`** of Flask-Login and gets the list of transactions from the user document in the database. Finally, the function returns the "Transactions.html" template with the list of transactions.

NOTES

The Project contains 3 .py files:

Datapopulation.py : it contains some functions that allow us to add Simulated Data to database so we can test the software

TransactionUtils.py : it contains some functions that works for transaction on blockchain with web3 library.

AffiliateTools.py: Here there are function that works for create the invited users chain and function that calculate the earnings.