Bright Money Assignment - Anand Kumar (kumar.annu0010@gmail.com)

In order to run the project run the following commands:

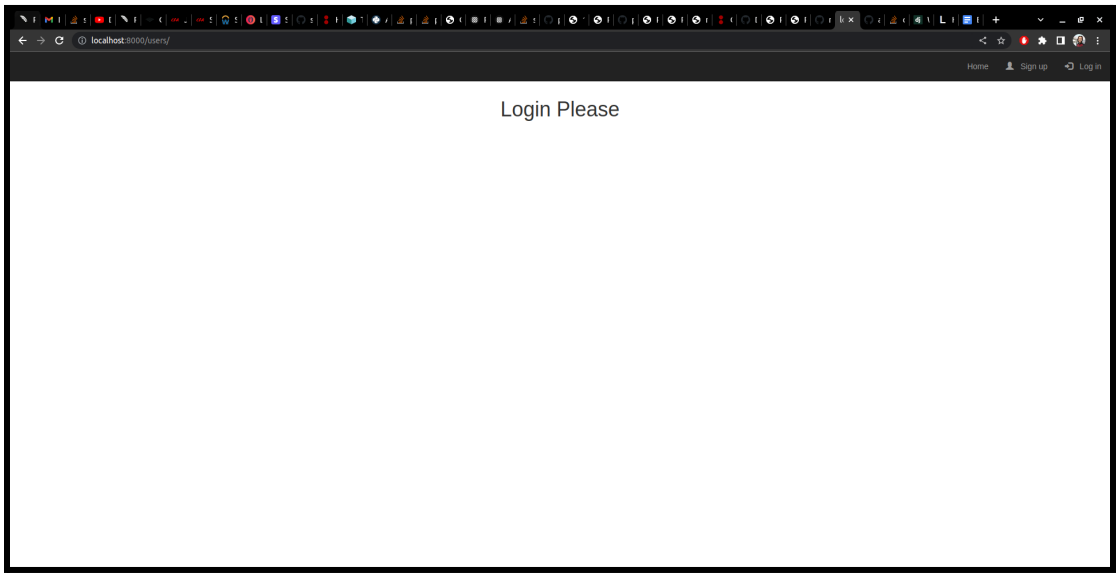
1) Create a virtual environment (using anaconda run **conda create -n plaidenv python=3.8** then activate using **conda activate plaidenv**)
2) Install dependencies by running **pip install -r requirements.txt**
3) **python manage.py makemigrations**
4) **python manage.py migrate**
5) **python manage.py runserver**
6) The website will we available at **http://localhost:8000/users/**

Following are the different steps involved in the project:

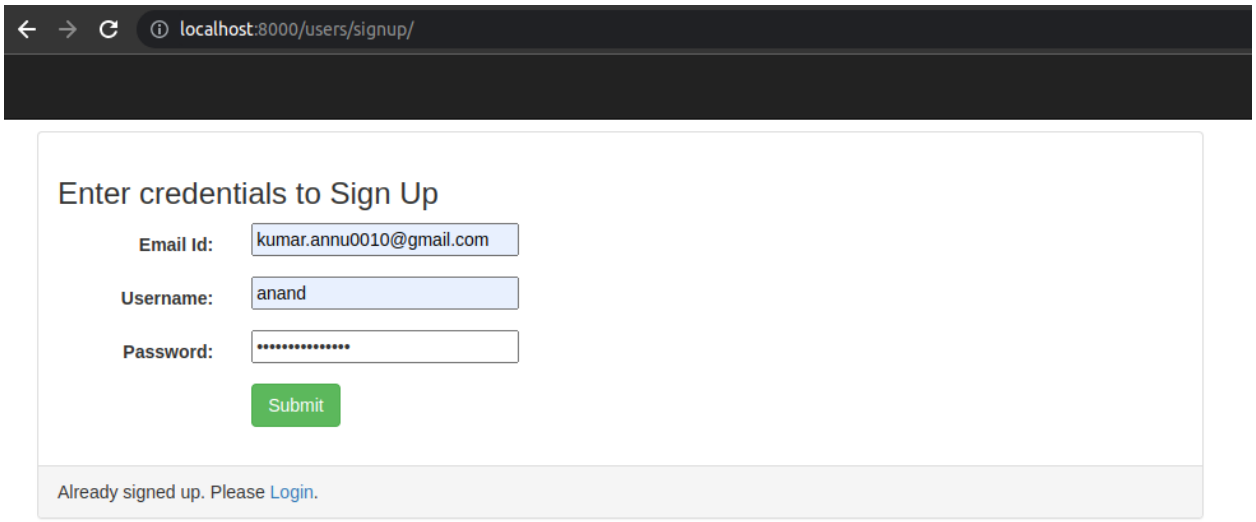
Step 1: Registering users

Created a class in **models.py** called **Users** with fields **username, password, email, is_logged_in, access_tkn and item_id** all of datatype character

Renders the **index.html.**



As a new user, click on signup. It renders signup.html and calls the action register in **views.py.**



register(request) gets the username, password, and email_id from the POST request from the HTML page.
Most API calls to Plaid endpoints require an access_token. An access_token provides access to a specific *Item*, which is a Plaid term for a login at a financial institution. To get the access_token we first need to generate a public key for the session.

This can be done by creating a client object of plaid and determining the institute id, and list of allowed products like transaction, balance, etc.

```
INSTITUTION_ID = 'ins_1' # The ID of the institution the Item will be
associated with
# Creating client
client = Client(client_id=PLAID_CLIENT_ID, secret=PLAID_SECRET_KEY,
environment=PLAID_ENV)
# Creating public token
res = client.Sandbox.public_token.create(
       INSTITUTION_ID,
       ['transactions'],
       webhook='https://sample-webhook-uri.com' # Expose a webhook for item
    )
Return res[‘public_token’]
```

Now we have the public token we can use this information to receive the access token and item_id to perform any Plaid API operation. We can use the legacy code to retrieve

the same. We are exposing a webhook for handling plaid transaction updates and fetch the transactions on receival of a webhook on website='https://sample-webhook-uri.com'

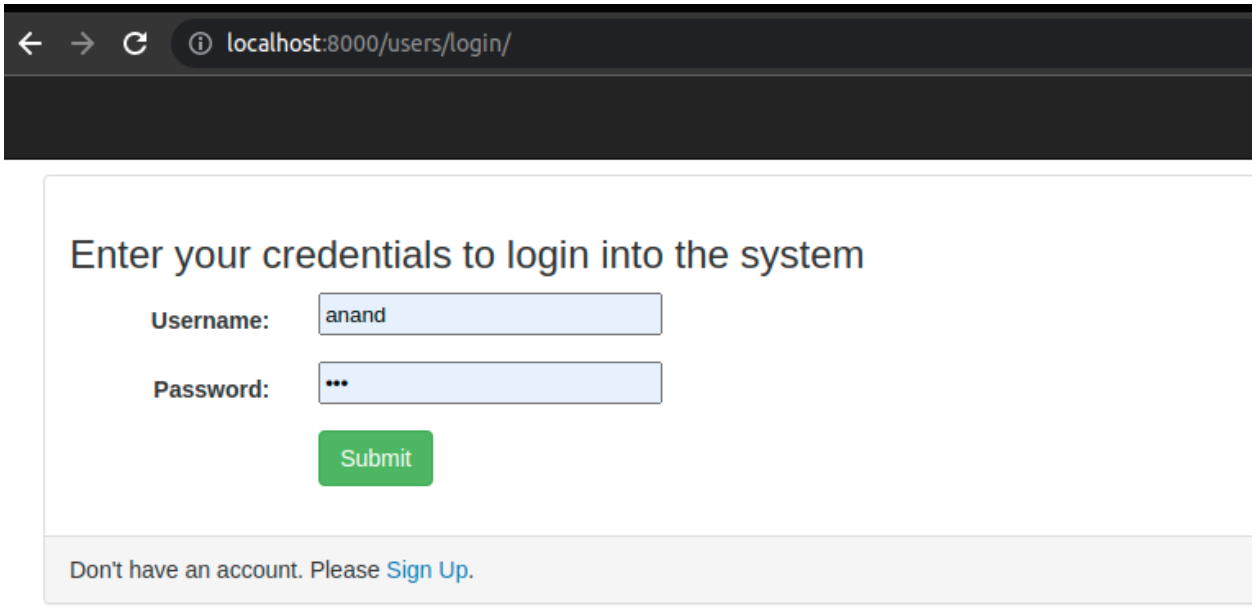
To get the access token and item_id we run the following code:

```
response = client.Item.public_token.exchange(public_token)
access_tkn = response['access_token']
item_id = response['item_id']
```

Now we have the access token and item id. We can associate these credentials with the user for validation and store it in db along other credentials that are email, username, and password. Thus the register API ends by storing the credentials in DB and template.render.

Step 2: Login Validation

In this step basically the user will have to input the username and password.



There is a validation function in views.py which gets the fields value. Then we check if the credentials are found in the db. If yes then we set **is_logged_in** of User **object = True**, and go to the profile page.
Else we raise a message with incorrect credentials and reload the login page.

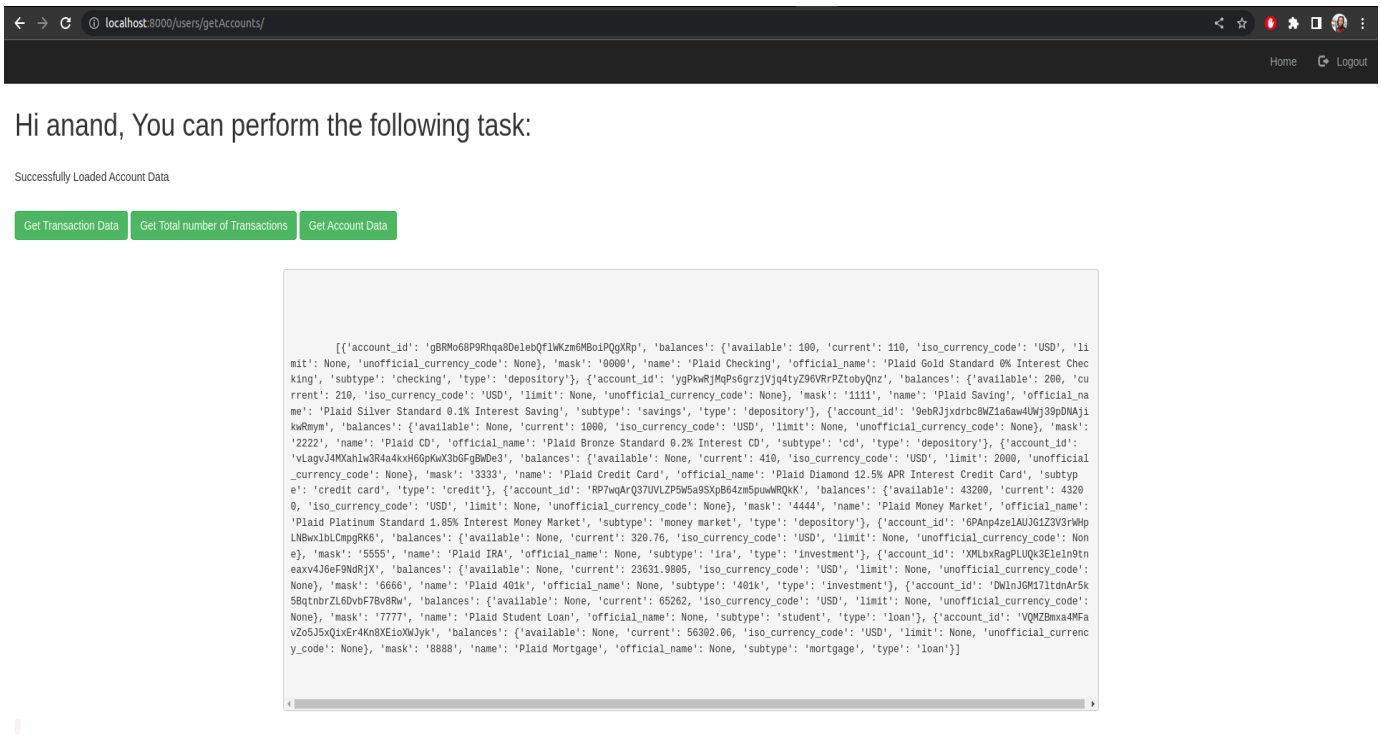
Step 3: Get Transaction and Account Detail

Currently 3 APIs have been integrated into the project.

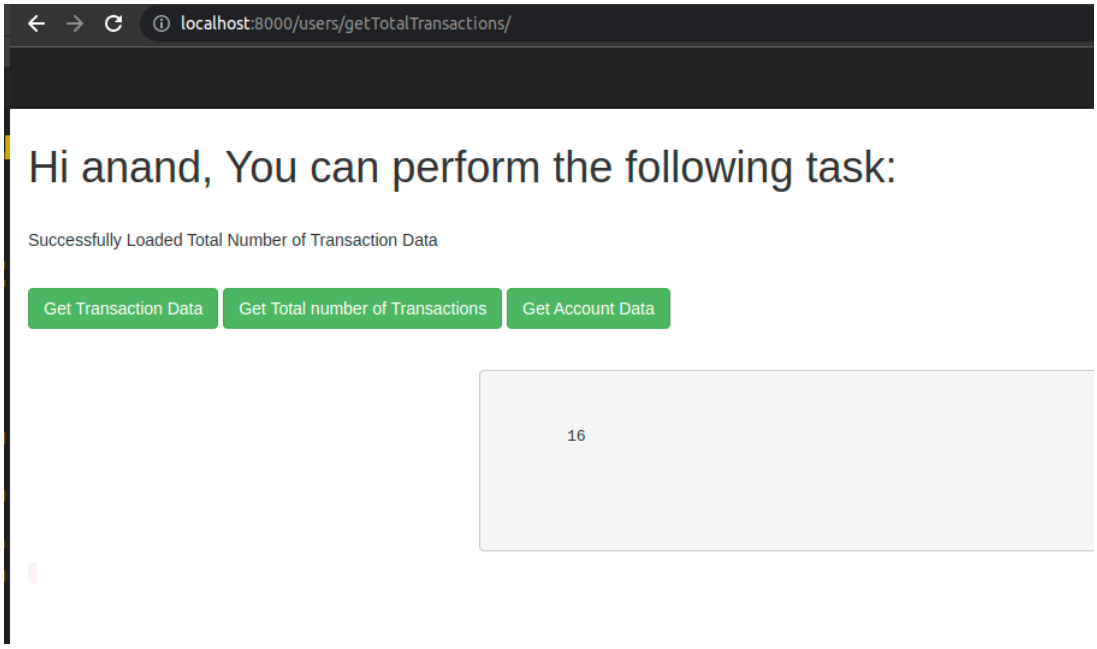
Transactions Data: Using the access token, we can specify the start date and end date between whose all transactions we want to get.

```
response = client.Transactions.get(access_tkn,
                                   start_date='2021-09-25',
                                   end_date='2021-10-25')
```

The response will return us 5 keys in JSON format - **'accounts', 'item', 'request_id', 'total_transactions', 'transactions'**. To get the transaction data we can simply **return response[''transactions'']**.



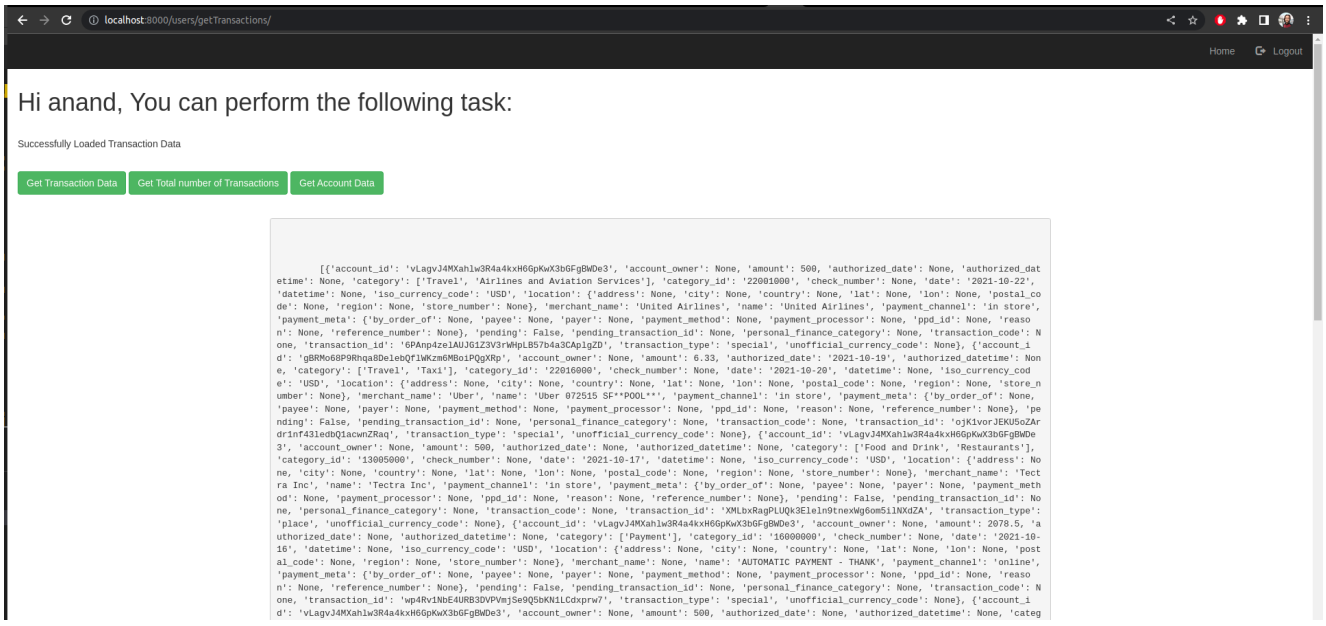
Total Number of Transactions: To get the total number of transactions we can simply **return response['total_transactions'].**



Account Data: Inorder to get the account data Plaid have an API for the same.

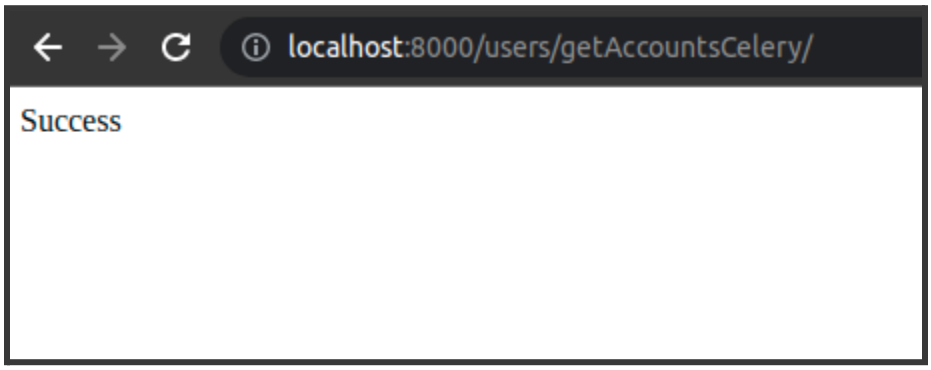
```
response = client.Accounts.balance.get(access_tkn,
                                       start_date='2021-09-25',
                                       end_date='2021-10-25')
```

The response will return 3 keys - **'accounts', 'item', 'request_id'**, we can return **response[‘accounts’]**

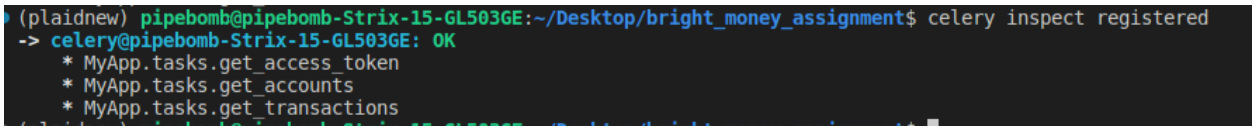


Step 4: Using celery for Asynchronous Task

We are using redis as the broker at port 6379. We create a task in task.py where we are getting the Accounts Data. An API is exposed with path getAccountsCelery. When it is called the access token is passed to the task and it gets the accountData, and returns Success.



When API is called the task is run via celery.

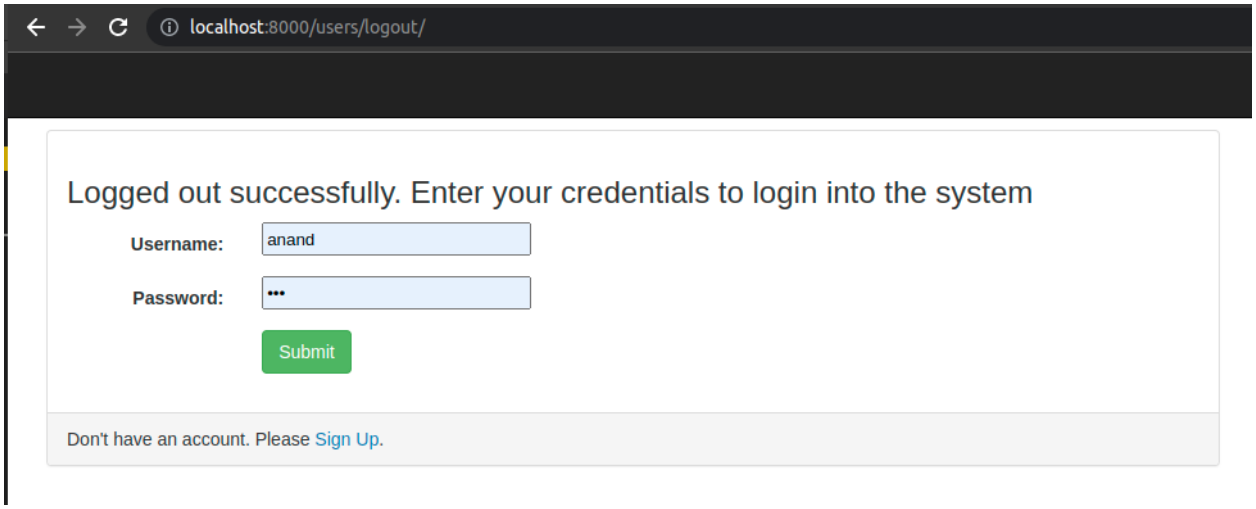


It shows the list of registered tasks.

To keep track of celery we can run `celery -A mainApp  worker -l info`
We can see all the current processes and their output. The above prints all the accountData details when the getAccountsCelery task is called.

Step 5: Logout

You can click on the logout button on the top right corner to logout. It will render the login page again.



Security of Credentials

The PLAID client ID, Secret Key and environment are stored as environment variables. These values are exported in bashrc script, and are loaded in the main folder settings.py file using os.getenv.

```
PLAID_CLIENT_ID = os.getenv('PLAID_CLIENT_ID')
PLAID_SECRET_KEY = os.getenv('PLAID_SECRET_KEY')
PLAID_ENV = os.getenv('PLAID_ENV')
```