Currency Exchange Application Documentation

Prepared by Dennis Njenga

For KOKO Networks Limited.

Currency Exchange Application

Overview

A Python-Flask Application that implements a currency exchange function using a currency conversion API that updates regularly.

Functional Attributes

1. The application intergrates a third party currency conversion service.

2. Creation of user accounts is enabled with a User Interface and comprises:

    a. Login/Sign-up

    b. Wallet

    c. Profile of user with profile photo, default currency of user which are both editable from the UI.

d. Credit and Debit functions using any currency and amount with automatic conversion depending on default currencies

e. Transfer money between users.

Technical Requirements

1. Application uses git to do progressive commits.

2. Uses a dynamic application architecture, pattern, models and quality.

3. Includes test cases.

4. Includes a README file with steps to preparation, installations and functionality of the application.

Currency Exchange Application

Preparation

mkdir currency-exchange

Git

git init

touch .gitignore

Installations

pip install virtualenv

Create Virtual Environment called ‘virtual’: virtualenv virtual

Activate Virtual Environment: source virtual/bin/activate

Install Flask: pip install flask

Install Requests: pip install requests

Bootstrap: pip install flask-bootstrap

Forms: pip install flask-wtf

Flash Script: pip install flash-script

Fonts: pip install Flask-FontAwesome

LogIn: pip install flask-login

Werkzeug: pip install -U Werkzeug==0.16.0

Database Installations: pip install flask-SQLAlchemy

pip install psycopg2

Database Migrations: pip install Flask-Migrate==2.7.0

python3 manage.py db init

python3 manage.py db migrate -m "Initial Migration"

python3 manage.py db upgrade

Enable easy run: chmod a+x start.sh

./start.sh

Currency Exchange Application

Development: Currency Exchange Function

Test Installation and the Virtual Environment

```
from flask import Flask, render_template, request
import requests
app = Flask(__name__)

@app.route("/")
def index():
    return "Hello World"

if __name__ == "__main__":
    app.run(debug=True)
```

API

The API chosen for this project is the Free Currency Converter. The API provider offers free web services for developers to convert one currency to another. Currency values are updated every 60 minutes but on the free plan may experience some downtime occasionally.

Test API functionality.

```
from flask import Flask, render_template, request
import requests
import os

app = Flask(__name__)

@app.route("/")
def index():
    API = os.environ.get("CURR_API")
    fromCurr = "USD"
    toCurr = "KES"
    url = f"https://free.currconv.com/api/v7/convert?
q={fromCurr}_{toCurr}&compact=ultra&apiKey={API}"
    rates = requests.get(url).json()
    return rates

if __name__ == "__main__":
    app.run(debug=True)
```

Currency Exchange Application

Input/Output Forms

Using Bootstap, create a input - output form to take in the currency data as input, process and provide the output as the exchange rate between two different currencies. For example USD to KES.

```
{%extends "base.html"%}
{%block body%}
    <form action="{{url_for('index')}}" method="post">
        <div class="form-group">
            <div class="form-group col-md-4">
                <label for="fromCurr">From:</label>
                <select id="fromCurr" class="form-control" name="fromCurr">
                    <option selected>Choose...</option>
                    <option></option>
                </select>
            </div>
        </div>
        <div class="form-group">
            <div class="form-group col-md-4">
                <label for="toCurr">To:</label>
                <select id="toCurr" class="form-control" name="toCurr">
                    <option selected>Choose...</option>
                    <option></option>
                </select>
            </div>
        </div>
        <div class="form-group">
            <input type="text" class="form-control" value="" disabled>
        </div>
        <button type="submit" class="btn btn-primary">Get Exchange
Rates</button>
    </form>
{%endblock%}
```

Retrieve Currency Data from API and format it in a way that will be utilized by the application.

```
if __name__ == "__main__":
    for data in requests.get("https://free.currconv.com/api/v7/currencies?apiKey={API}").json()["results"]:
        print(data)
```

Enable conversion rate from any known currency to any other know currency by display the exchange rate/ For example From USD to KES the exchange rate displayed is 107.7949 KES.

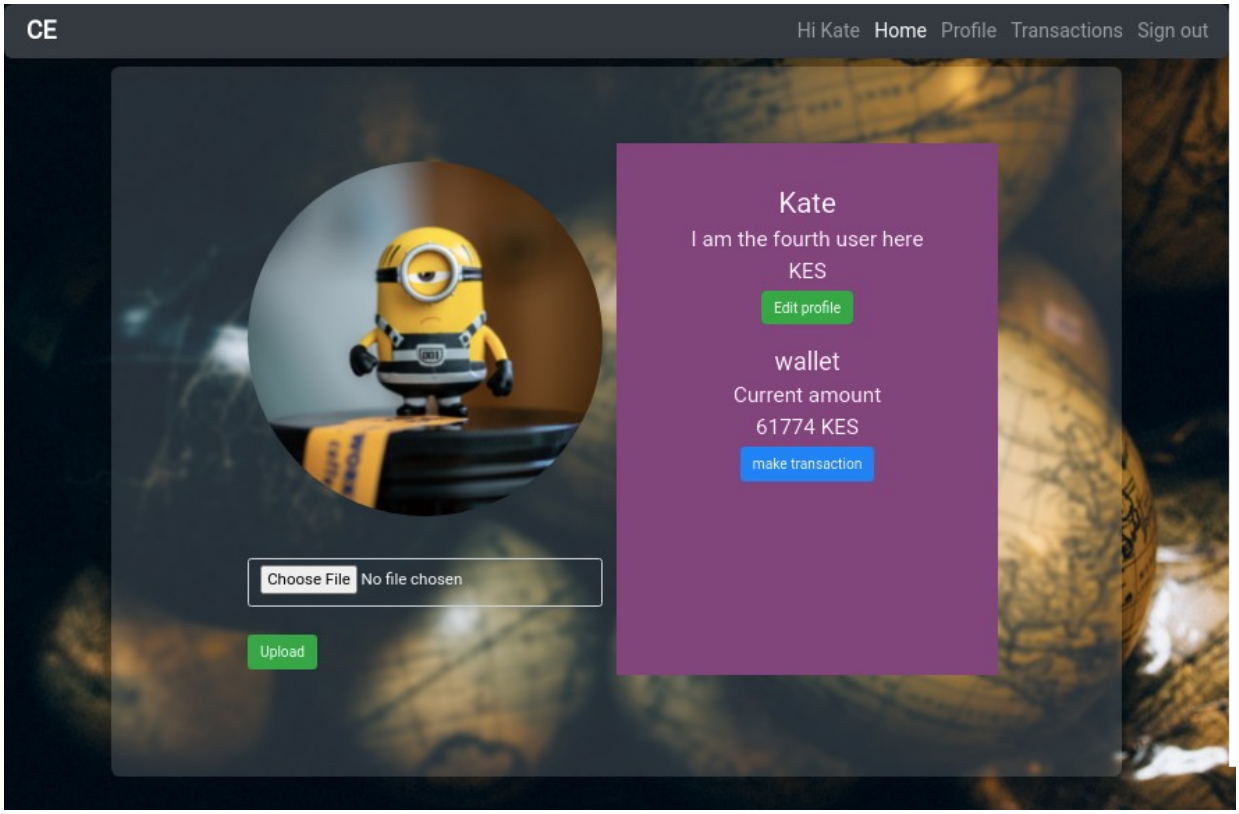
```
@app.route("/", methods=["POST", "GET"])
def index():
    if request.method == "POST":
        API = os.environ.get("CURR_API")
        fromCurr = request.form.get('fromCurr')
        toCurr = request.form.get('toCurr')
        if fromCurr and toCurr:
            try:
                url = f"https://free.currconv.com/api/v7/convert?q={fromCurr}_{toCurr}&compact=ultra&apiKey={API}
                rates = requests.get(url).json()[f'{fromCurr}_{toCurr}']
                return render_template('index.html', currency=currencys, result=f"{rates} {toCurr}")
            except:
                flash("Error Occured", 'danger')
                return redirect(url_for('index'))
        else:
            flash("Error Occured", 'danger')
            return redirect(url_for('index'))

    return render_template('index.html', currency=currencys)
```

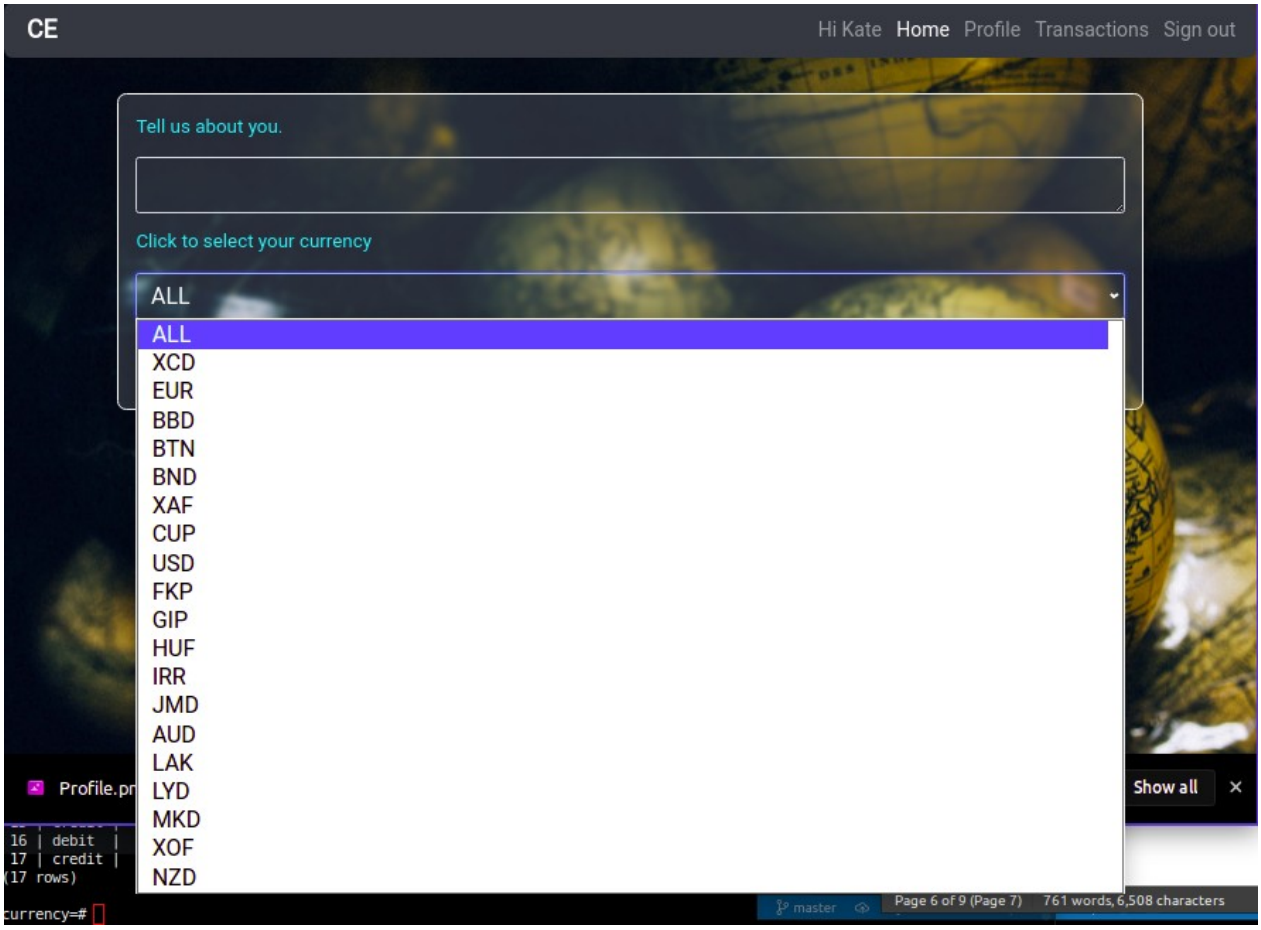
User Authentication

Authentication comprises a log-in, signup, profile and edit profile function that includes the option to update a profile picture and select a default currency. Every user starts with a KES 1000 balance converted into their default currency.

Currency Exchange Application



Edit Profile



Currency Exchange Application

Database - Tables

```
             List of relations
 Schema |      Name       | Type  |   Owner
--------+-----------------+-------+-----------
 public | alembic_version | table | deepeters
 public | transactions    | table | deepeters
 public | users           | table | deepeters
 public | wallets         | table | deepeters
(4 rows)
```

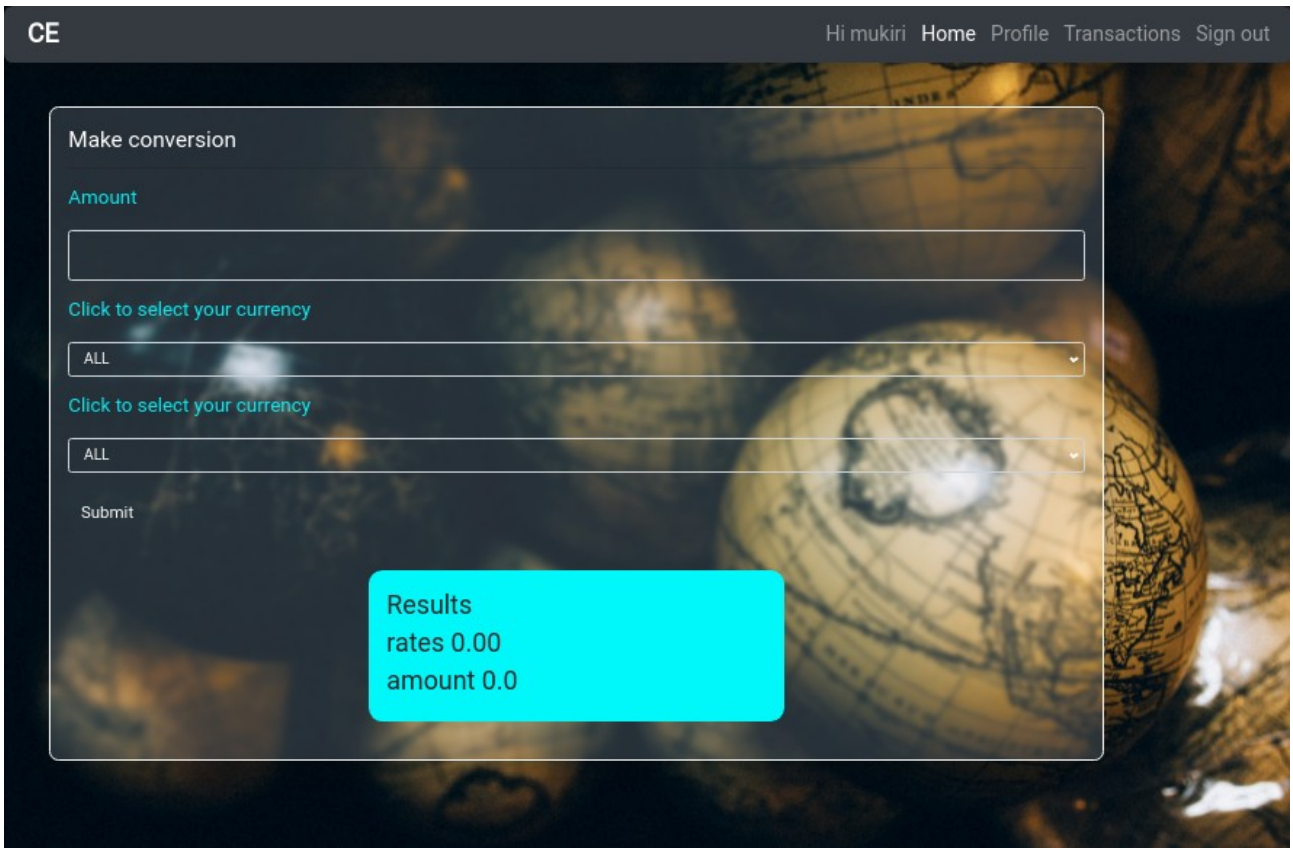
Sample Transactions Table

```
currency=# SELECT * FROM transactions;
 id |  type  | amount |            time            | user_id | wallet_id
----+--------+--------+----------------------------+---------+-----------
  1 | credit |   1000 | 2021-06-16 23:42:36.050322 |       1 |         1
  2 | credit |   1000 | 2021-06-16 23:49:54.159081 |       2 |         2
  3 | debit  |  20000 | 2021-06-16 23:51:51.133788 |       2 |         2
  4 | credit |  40000 | 2021-06-16 23:52:23.188492 |       2 |         2
  5 | debit  |    100 | 2021-06-16 23:52:49.632439 |       2 |         2
  6 | credit |  10785 | 2021-06-16 23:52:50.777005 |       1 |         1
  7 | credit |   1000 | 2021-06-16 23:55:41.480306 |       3 |         3
  8 | debit  |    100 | 2021-06-17 00:10:25.037367 |       3 |         3
  9 | credit |    120 | 2021-06-17 00:10:25.984903 |       2 |         2
 10 | debit  |  15000 | 2021-06-17 00:10:53.475266 |       2 |         2
 11 | credit |  12508 | 2021-06-17 00:10:54.097378 |       3 |         3
 12 | credit |   1000 | 2021-06-17 00:12:56.846257 |       4 |         4
 13 | credit |   1000 | 2021-06-17 00:14:44.236454 |       4 |         4
 14 | credit |   2000 | 2021-06-17 00:14:58.460427 |       4 |         4
 15 | credit |   1000 | 2021-06-17 19:30:01.284684 |       4 |         4
 16 | debit  |  50000 | 2021-06-17 19:30:49.947934 |       4 |         4
 17 | credit |  50000 | 2021-06-17 19:30:50.636809 |       1 |         1
(17 rows)
```

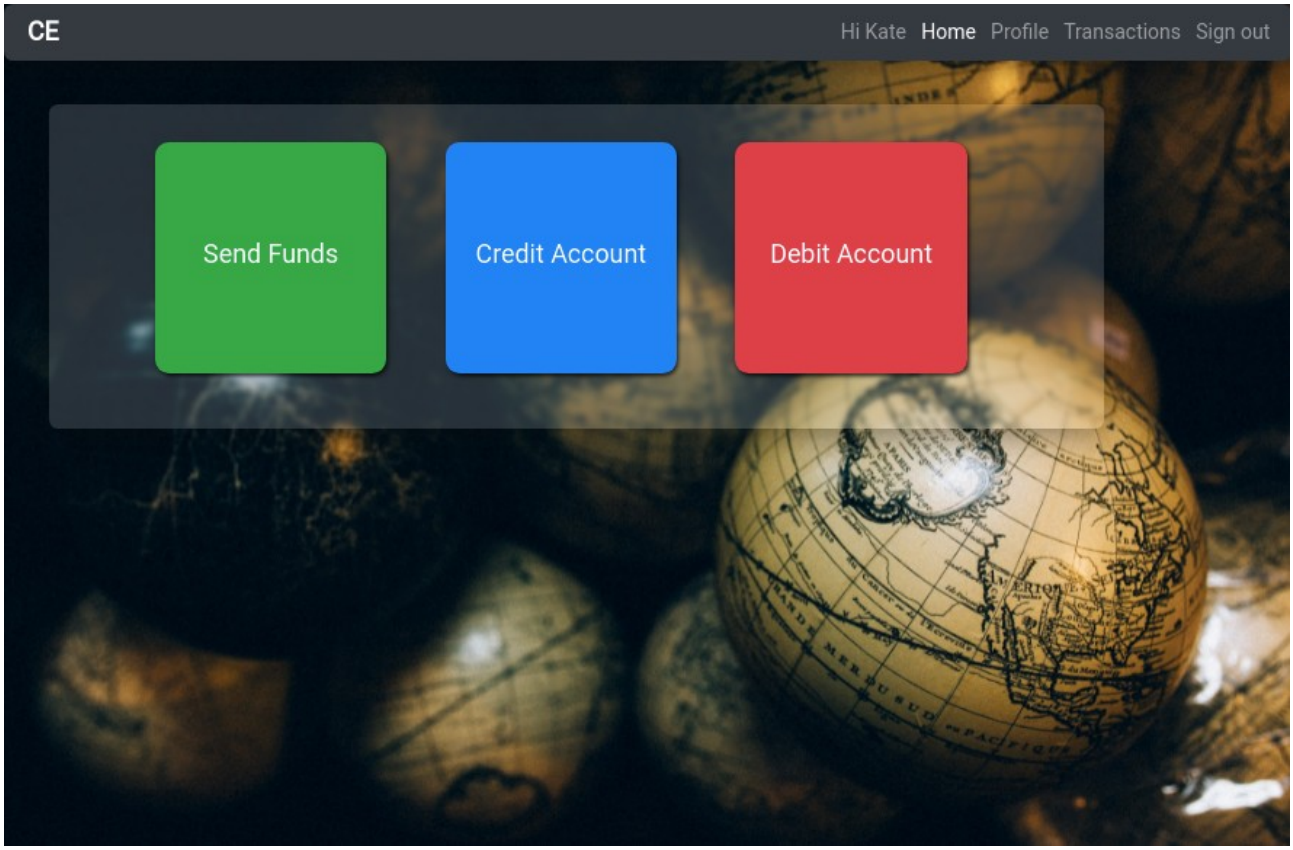
Currency Exchange Application

Currency Conversion

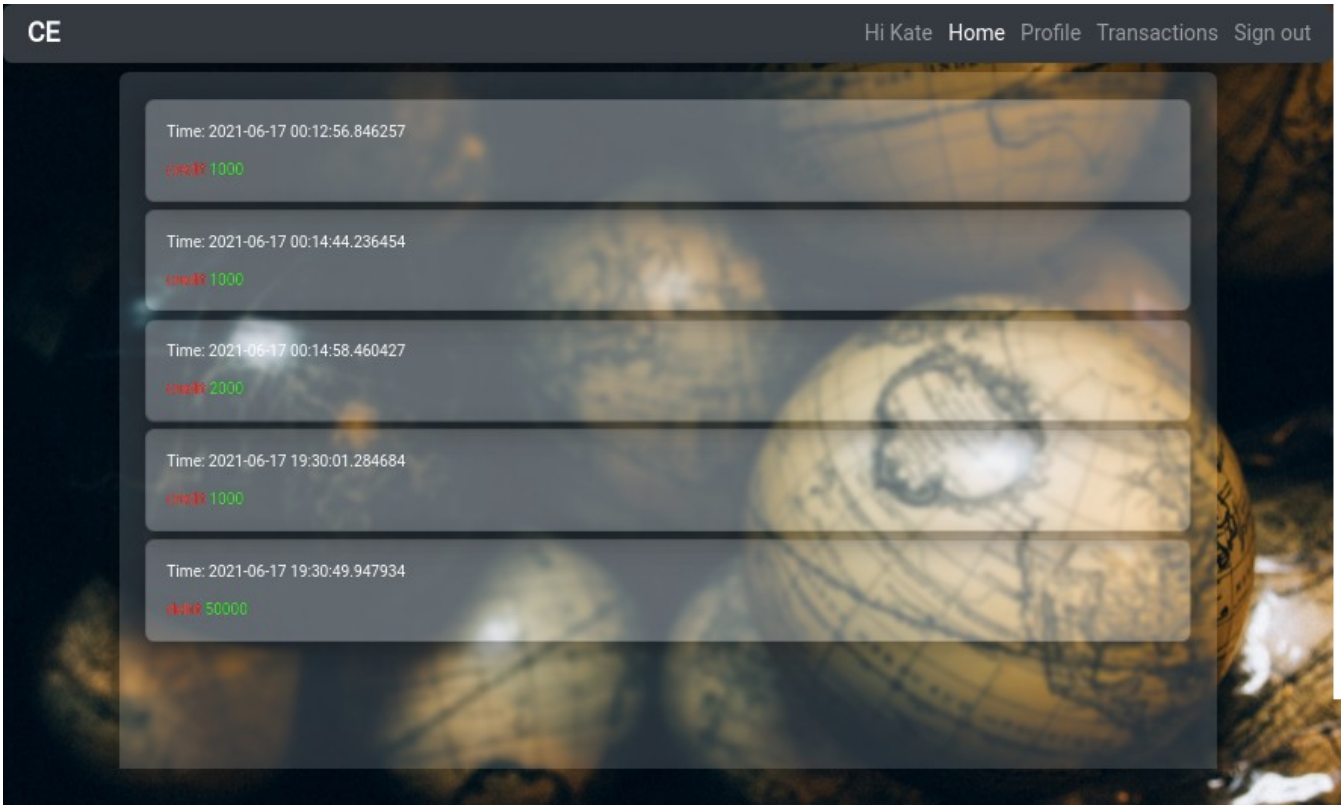
A user can convert from one currency to the other on request using the “Make Conversion” function. Another way conversion happens is through transactions. For instance, when one user sends money to another user in the same app, in the case that their default currencies are different, conversion happens by default and reaches the receiver wallet in their default currency. The conversions are up to date with the conversion rates which update every hour. Debit and Credit functions can also be performed in any currency by a user to their own wallets and the money is saved to their wallets in their individual currencies.



Currency Exchange Application



Transactions are displayed on the transactions page depending on the user where they can view debits and credits that occurred in their accounts as statement of accounts. Time and amount are also included in the report.



Currency Exchange Application

Deployment

Deployment has been achieved through Heroku. Heroku is a cloud deployment platform that is able to implement both the application dependencies and the database.

1. pip install gunicorn

2. pip freeze > requirements.txt

3. touch Procfile and write the code: web: gunicorn manage:app into the file.

4. heroku login

5. heroku create currency-exchange-application

6. heroku config:set MOVIE_API_KEY=<YOUR MOVIE API>

7. heroku config:set SECRET_KEY=<YOUR SECRET KEY>

8. heroku addons:create heroku-postgresql

9. Alter config.py with:

    class ProdConfig(Config):

    SQLALCHEMY_DATABASE_URI = os.environ.get("DATABASE_URL")

10. Alter manage.py

    From: app = create_app(‘development’)

    To: app = create_app(‘production’)

11. pip freeze > requirements.txt

12. Push to App Heroku

    git push heroku master

13. Deploy Database

    heroku run python3 manage.py db upgrade

Suggestions as the App expands

As the business grows, the number of requests is expected to increase on the server. To handle an increased number of requests, the application would need to use another API as the one applied here has a limit on the number of responses and requests it can provide and receive per unit time. The future applied API also needs to update the exchange rates instantly as in the exchange market rather than the 60 minute rate provided in this application.