**Phase 1 Source Code**

Currency Conversion Application

UMGC CMSC 495 Section 6380

Elizabeth Bloss

Jackson Perry

Carl Blocker

Jonah Kiplimo

**Introduction**

Group 1’s project is to develop a currency conversion web application where the user will enter information (initial value, base currency, desired conversion currency) and the converted monetary value will be displayed.

**Objective**

The objective of this project is to allow users to convert form one currency to another in near real time.

* Support popular global currencies (USD, EUR, GBP, INR, JPY, etc.).
* Display accurate and up-to-date exchange rates.
* Ensure easy usability on both desktop and mobile devices.
* Provide a secure and fast user experience to retain users.

**Instructions**

**Project Setup:**

The initial project setup consisted of adding a project called CMSC/495 to GitHub and creating a private repository with all group members. This is to serve as the version control system. This also manages secrets and is linked to a container registry to deploy the main Flask app in the GitHub container registry for easy version control and deployment.

Next each member had to set up VS Code, the IDE every member is using, and created the virtual environment and .env files on each client. Each member created a clean virtual environment with Python 3.12 and then built from a requirements.txt synced via GitHub. Currently the requirements are:

astroid==3.3.10

blinker==1.9.0

certifi==2025.4.26

charset-normalizer==3.4.2

click==8.2.1

dill==0.4.0

Flask==3.1.1

Flask-SQLAlchemy==3.1.1

git-filter-repo==2.47.0

greenlet==3.2.2

gunicorn==23.0.0

idna==3.10

iniconfig==2.1.0

isort==6.0.1

itsdangerous==2.2.0

Jinja2==3.1.6

MarkupSafe==3.0.2

mccabe==0.7.0

packaging==25.0

platformdirs==4.3.8

pluggy==1.6.0

psycopg2-binary==2.9.10

Pygments==2.19.1

pylint==3.3.7

pytest==8.4.0

pytest-flask==1.3.0

python-dotenv==1.1.0

requests==2.32.3

SQLAlchemy==2.0.41

tomlkit==0.13.2

typing\_extensions==4.13.2

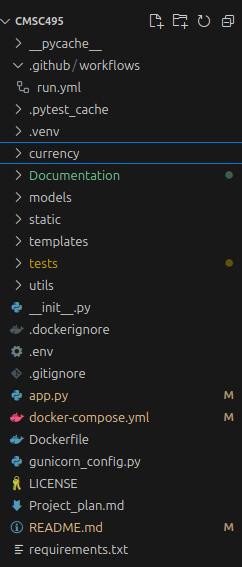
urllib3==2.4.0

Werkzeug==3.1.3

**Structure of Software System (some files omitted for clarity):**

├── app.py

├── currency

│ ├── client.py

│ ├── \_\_init\_\_.py

├── docker-compose.yml

├── Dockerfile

├── .dockerignore

├── Documentation

│ ├── ArchitectureDiagram.svg

│ ├── Group 1 Phse 1 source.docx

│ ├── Group 1 Project Design.docx

│ ├── packages.png

│ └── Project Plan Group 1 (4).docx

├── .env

├── .git

\*truncated

├── .github

│ └── workflows

├── .gitignore

├── gunicorn\_config.py

├── \_\_init\_\_.py

├── LICENSE

├── models

│ ├── \_\_init\_\_.py

│ ├── models.py

│ └── \_\_pycache\_\_

├── Project\_plan.md

├── README.md

├── requirements.txt

├── static

│└── countries.json

├── templates

│ ├── index.html

│ └── partials

├── tests

│ ├── conftest.py

│ ├── \_\_intit\_\_.py

│ ├── test\_client.py

│ ├── test\_logging.py

│ └── test\_routes.py

├── utils

│ ├── create\_flag\_png.py

│ ├── currency\_logging.py

│ ├── \_\_init\_\_.py

│ └── seed\_countries\_db.py

└── .venv

\*truncated

**Implementation of Software:**

The most important file to get started is the .env file. It presumes a Postgres database with a currency\_logs database. Setting that up is described in the README.md file and it should follow the format below.

**# .env file**

**app\_id=<app\_id from openexchangrates.org>**

**DEBUG=true**

**FLASK\_ENV=development**

**pg\_password=<strongPassword>**

**pg\_user=currencyuser**

**pg\_ip=xxx.xxx.xx.xxx:5432**

Once this is created correctly, the rest of the repository can be cloned from GitHub.

**/currency/client.py**

# currency/client.py

import requests

import base64

from models.models import Country

class CurrencyClient:

"""this class makes the connection to the API and has methods to convert currencies"""

def \_\_init\_\_(self, app\_id):

self.app\_id = app\_id

self.currency\_list\_url = "https://openexchangerates.org/api/currencies.json"

self.base\_currency = "USD"

self.target\_currency = "EUR"

self.currency\_choices = {"USD": "US Dollar", "EUR": "Euro"}

self.currency\_flags = {}

self.base\_value = 0

# Do NOT call this by default

# self.get\_currency\_choices\_from\_db()

def set\_base\_value(self, value):

"""sets the base value for the currency being converted from"""

self.base\_value=value

def get\_currency\_choices\_from\_db(self):

"""gets the currency choices from the PG db"""

db\_choices = {}

for country in Country.query.all():

flag\_base64 = base64.b64encode(country.flag).decode('utf-8')

db\_choices[country.currency\_code] = country.currency\_name

self.currency\_flags[country.currency\_code] = flag\_base64

self.currency\_choices=db\_choices

def get\_currency\_choices(self):

"""retrieves all the currency choices avaible at the endpoint"""

response = requests.get(self.currency\_list\_url, timeout =8)

if response.status\_code ==200:

data = response.json()

self.currency\_choices = data

else:

raise ValueError("Expected a JSON object (dict) from /currencies")

def set\_base\_currency(self,base):

"""sets the base currency"""

self.base\_currency=base

def set\_target\_currency(self,target):

"""sets the target currency"""

self.target\_currency= target

#this is only allowing USD base at the free subscription

#fixed calls dollars to both currency and uses divison

def calculate(self):

"""calcultes the conversion currenly only working for USD base"""

url=f"https://openexchangerates.org/api/latest.json?app\_id={self.app\_id}&base=USD&symbols={self.base\_currency},{self.target\_currency}&prettyprint=false&show\_alternative=false"

response = requests.get(url, timeout =8)

if response.status\_code ==200:

raw\_data =response.json()

data=(raw\_data["rates"][self.target\_currency]/raw\_data["rates"][self.base\_currency])\*self.base\_value

return round(data,2)

else:

return response.status\_code

**/models/models.py**

from datetime import datetime, timezone

from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class VisitorLog(db.Model):

\_\_tablename\_\_ = 'visitor\_logs'

id = db.Column(db.Integer, primary\_key=True)

timestamp = db.Column(db.DateTime(timezone=True), default=datetime.now(timezone.utc))

ip\_address = db.Column(db.Text)

user\_agent = db.Column(db.Text)

referrer = db.Column(db.Text)

method = db.Column(db.Text)

path = db.Column(db.Text)

event\_type = db.Column(db.Text) # 'visit' or 'conversion'

base\_currency = db.Column(db.Text)

target\_currency = db.Column(db.Text)

amount = db.Column(db.Numeric)

converted = db.Column(db.Numeric)

class Country(db.Model):

\_\_tablename\_\_ = 'countries'

id = db.Column(db.Integer, primary\_key=True, autoincrement=True)

name = db.Column(db.String, nullable=False)

iso\_alpha2 = db.Column(db.String(2), nullable=False)

currency\_name = db.Column(db.String, nullable=False)

currency\_code = db.Column(db.String(3), nullable=False)

currency\_symbol = db.Column(db.String(10), nullable=False)

flag = db.Column(db.LargeBinary, nullable=False) # Stores binary flag image

**/app.py**

import os

import base64

from datetime import datetime, timezone

from collections import deque

from flask import Flask, request, render\_template

from dotenv import load\_dotenv

from flask import g

from models.models import db, Country

from utils.currency\_logging import log\_event

from currency.client import CurrencyClient

def create\_app(test\_config=None):

# Only load .env file when not in production

if os.getenv('FLASK\_ENV') != 'production':

load\_dotenv()

app = Flask(\_\_name\_\_)

# Load default or test config

if test\_config is None:

app\_id = os.getenv('app\_id')

pg\_password = os.getenv("pg\_password")

pg\_user = os.getenv('pg\_user')

pg\_ip = os.getenv('pg\_ip')

DATABASE\_URL = f"postgresql://{pg\_user}:{pg\_password}@{pg\_ip}/currency\_logs"

app.config["SQLALCHEMY\_DATABASE\_URI"] = DATABASE\_URL

else:

app.config.update(test\_config)

app\_id = test\_config.get("APP\_ID", "test")

app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = False

db.init\_app(app)

client = CurrencyClient(app\_id)

conversion\_history = deque(maxlen=10)

def get\_countries():

"""retrieve and cache countries from database"""

if 'countries' not in g:

try:

g.countries = Country.query.order\_by(Country.name).all()

except Exception as e:

app.logger.warning(f"Could not retrieve countries: {e}")

g.countries = []

return g.countries

@app.before\_request

def handle\_before\_request():

"""inital setup of client session before users pick anything"""

if not hasattr(g, "\_currency\_preloaded"):

try:

client.get\_currency\_choices\_from\_db()

except Exception as e:

app.logger.warning(f"Failed to preload currency data: {e}")

g.\_currency\_preloaded = True

if request.method == "GET" and request.path == "/":

log\_event(event\_type="visit")

@app.teardown\_appcontext

def shutdown\_session(exception=None):

"""Clean up the database session after each request."""

db.session.remove()

@app.route("/test")

def test\_route():

"""simple test route to see if server is up"""

print("Test route accessed")

return "Test successful"

@app.route("/", methods=["GET", "POST"])

def home():

"""

Renders the currency converter form and handles form submissions.

- On GET: Displays the form for selecting base/target currencies and entering an amount.

- On POST: Processes the form data, performs the currency conversion using the API,

and displays the result or any error that occurs.

Returns:

A rendered HTML page (index.html) with the conversion form, result, and optional error message.

"""

result = None

error = None

#currencies = client.currency\_choices

#flags = client.currency\_flags

# Use database for dropdown labels and flags

currency\_client = client

countries = get\_countries()

currency\_options = [

{

"code": c.currency\_code,

"label": f"{c.name} {c.currency\_name} ({c.currency\_code})",

"flag": c.flag

}

for c in countries

]

flags = {

c.currency\_code: base64.b64encode(c.flag).decode("utf-8")

for c in countries

}

if request.method == "POST":

try:

base = request.form["base\_currency"]

target = request.form["target\_currency"]

value = float(request.form["amount"])

currency\_client.set\_base\_currency(base)

currency\_client.set\_target\_currency(target)

currency\_client.set\_base\_value(value)

converted=currency\_client.calculate()

result = f"{value} {currency\_client.base\_currency} is {converted} {currency\_client.target\_currency}"

#log events

try:

log\_event(

event\_type="conversion",

base=base,

target=target,

amount=value,

result=converted

)

except Exception:

error = "⚠️ We were unable to log your activity, but your conversion was successful."

# Save query to history

conversion\_history.appendleft({

"time": datetime.now(timezone.utc).strftime("%Y-%m-%d %H:%M:%S"),

"base": base,

"target": target,

"amount": value,

"result": result

})

# clean up the exceptions needed

except Exception as e:

error = str(e)

return render\_template("index.html", currencies=currency\_options, flags=flags, result=result, error=error, history=conversion\_history)

@app.route("/flag-preview", methods=["POST"])

def flag\_preview():

"""create a flag icon for the source currencyfrom base64 encoded databse entry"""

currency\_client = client

code = request.form.get("base\_currency", "USD")

country = next((c for c in get\_countries() if c.currency\_code == code), None)

flags = currency\_client.currency\_flags

flag = flags.get(code, "")

if country:

label = f"{country.name} {country.currency\_name}"

else:

label = f"Unknown ({code})"

return render\_template("partials/flag\_preview.html", code=code, name=label, flag=flag)

@app.route("/target-flag-preview", methods=["POST"])

def target\_flag\_preview():

"""create a flag icon for the target currencyfrom base64 encoded databse entry"""

currency\_client = client

code = request.form.get("target\_currency", "EUR")

country = next((c for c in get\_countries() if c.currency\_code == code), None)

flags = currency\_client.currency\_flags

flag = flags.get(code, "")

if country:

label = f"{country.name} {country.currency\_name}"

else:

label = f"Unknown ({code})"

return render\_template("partials/flag\_preview.html", code=code, name=label, flag=flag)

return app

if \_\_name\_\_ == "\_\_main\_\_":

app = create\_app()

with app.app\_context():

db.create\_all()

app.run(debug=True, port=5001)

**/gunicorn\_config.py**

import os

workers = int(os.environ.get('GUNICORN\_PROCESS', '2'))

threads= int(os.environ.get('GUNICORN\_THREADS', '4'))

bind = os.environ.get('GUNICONR\_BIND', '0.0.0.0:8080')

forwarded\_allow\_ips ='\*'

secure\_scheme\_headers = {'X-Forwarded-Proto': 'https'}

**Testing the Software:**

Software testing is done formally with Pytest. Three files were created to test the basic functionality of the software in isolation of the API requests. The group intends to add more assertions for more robust testing in the upcoming week.

**/tests/conftest.py**

import os

import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..")))

import pytest

from app import app as flask\_app

from models.models import db

@pytest.fixture

def app():

flask\_app.config.update({

"TESTING": True,

"SQLALCHEMY\_DATABASE\_URI": "sqlite:///:memory:",

"SQLALCHEMY\_TRACK\_MODIFICATIONS": False,

})

with flask\_app.app\_context():

db.create\_all()

yield flask\_app

db.session.remove()

db.drop\_all()

@pytest.fixture

def client(app):

return app.test\_client()

**/tests/test\_client.py**

from currency.client import CurrencyClient

def test\_currency\_client\_calculate(monkeypatch):

dummy\_response = {

"rates": {"USD": 1.0, "EUR": 0.9}

}

def mock\_get(url, timeout):

class Dummy:

def \_\_init\_\_(self):

self.status\_code = 200

def json(self):

return dummy\_response

return Dummy()

monkeypatch.setattr("requests.get", mock\_get)

client = CurrencyClient("fake\_app\_id")

client.set\_base\_currency("USD")

client.set\_target\_currency("EUR")

client.set\_base\_value(100)

result = client.calculate()

assert result == 90.0

**/tests/test\_logging.py**

from utils.currency\_logging import log\_event

def test\_log\_event\_gets\_added(app, client):

with app.test\_request\_context("/", headers={"User-Agent": "test"}, environ\_base={"REMOTE\_ADDR": "127.0.0.1"}):

log\_event(event\_type="test", base="USD", target="EUR", amount=100, result=90)

from models.models import VisitorLog

log = VisitorLog.query.first()

assert log is not None

assert log.event\_type == "test"

assert log.base\_currency == "USD"

/tests/test\_routes.py

def test\_home\_page\_loads(client):

response = client.get("/")

assert response.status\_code == 200

assert b"Currency" in response.data or b"form" in response.data

def test\_test\_route(client):

response = client.get("/test")

assert response.status\_code == 200

assert b"Test successful" in response.data

A black screen with green and white lines

AI-generated content may be incorrect.

**Conclusion**

The Phase I project source code outlines the Python code required to develop a Flask app that can convert currency pairs. Also attached as an appendix is the git archive file to assist in setting up the containers and databases required to deploy the container. There are currently three test cases written in Pytest that test each of the three primary modules in the code. There are several open issues that still need to be resolved including cleaning some data in the database seed and front end design.