**CIS 2286. Lab Assignment 4.**

**Classes**

Goal:

Re-write the application using python classes. Add support for transactions storage. Add access options to display transactions (all and filtered by identifier) and create support for these new actions. Calculate and display the total profit for stored transactions.

Checkout branch lab4.

You do not need to modify any code you checked out from git, except these cases:

1. Add code in “TODO” commented section
2. Replace function body “pass” statement by function implementation.

***Detailed requirements:***

# Stage1.

**quote\_model.py** and **quote.py** are converted to classes and re-written

This part is already done in the initial code for lab4.

# Stage2.

1. Re-write **transaction.py** as classes Transaction, TransactionList, TotalIncome

This part was prototyped for you. Please note that exchange function was renamed into post method and moved to TransactionList class , some methods were made private. You may want to use **quote.py** as your guide.

1. Implement transaction storage **transaction\_model.py** as Python class.
2. Add class attributes:

**transaction\_list** to store transactions similar to lab2 list for quotes

**id** transaction id generator : incremented by 1 for each consequent transaction

**total** as sum of all transactions commissions and transaction

**total = commission1 + commission2 + ..**

**commission1 = commission\_rate \* transaction\_amount\_in\_USD**

**commission\_rate = 0.1 (10%)**

**commission is calculates in USD**

For example, transaction of 85 EUR will have 10 USD commission given conversion\_rate= 1.17

# Stage3.

Add support for the following actions**: O, A, T**

* display one transaction

**action=’O’**

values to collect from user:

**identifier (positive integer)**

* display all transactions

**action=’A’**

* display sum of commissions for all stored transactions

**action=’T’**

# Stage4.

***Test use cases:***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **action** | **currency\_from** | **currency\_to** | **amount** | **conversion\_rate** | **id** |
| 1 | Q | USD | EUR | ---- | ---- | ---- |
| 2 | S | USD | GBP | ---- | 0.74 | ---- |
| 3 | D | ---- | ---- | ---- | ---- | ---- |
| 4 | X | GBP | AUD | 1000 | ---- | ---- |
| 5 | X | AUD | EUR | 1000 | ---- | ---- |
| 6 | X | AUD | GPB | 1000 | ---- | ---- |
| 7 | S | EUR | AUD | ---- | 1.59 | ---- |
| 8 | D | ---- | ---- | ---- | ---- | ---- |
| 9 | A | ---- | ---- | ---- | ---- | ---- |
| 10 | T | ---- | ---- | ---- | ---- | ---- |
| 11 | O | ---- | ---- | ---- | ---- | 1 |
| 12 | O | ---- | ---- | ---- | ---- | 3 |
| 13 | O | ---- | ---- | ---- | ---- | 7 |

***Code structure for lab4***

The project code contains several files with the new function prototypes for Lab4 solution. The goal is to implement these methods.

**currency-exchange**

**├── \_\_init\_\_.py**

**├── app.py**

**├── common**

**│   ├── \_\_init\_\_.py**

**│   └── utils.py**

**├── log**

**├── models**

**│   ├── \_\_init\_\_.py**

**│   ├── quote\_model.py**

**│   └── transaction\_model.py**

**├── resources**

**│   ├── \_\_init\_\_.py**

**│   ├── quote.py**

**│   └── transaction.py**

**└── tests**

**├── \_\_init\_\_.py**

**└── currency\_exchange\_tests.py**

The structure of this tree is designed for the future move from a command line script to a Flask application. We will start outlining our project as a RESTful application from the beginning.

**currency-exchange/** is the root directory for this project and the name of the service (application)

**app.py** is the main entry point of the application. The logic is outlined in app.main\_command\_line()

**common**/ consists of common classes and functions that are useful for all the modules. Some examples are global constants, global helper functions or logging utilities. Currently we have util.py module with log function

**resources/** - collection of modules representing resources. Resource is the fundamental concept in any RESTful application. A resource is a unit of a REST API that you can call. It has some type, associated data, methods operating on it and relationships with other resources. In our project we will be defining two resources:

- **quote**: an exchange rate between two currencies

- **transaction**: a transfer of a given amount from one currency to another

**\_\_init\_\_.py** - files are required to make Python treat the directories as containing packages; this is done to prevent directories with a common name, such as string, from unintentionally hiding valid modules that occur later (deeper) on the module search path

**tests/currency\_exchange\_tests.py -** unit tests

**models/** - collection of modules representing data storage.