**CIS 2286. Lab Assignment 3.**

**Lists**

**Dictionaries**

**Classes**

Goal:

Part1. Refactor and clean the code

Part2. Implement currency quotes cache storage through Python data structure.

Checkout branch lab3.

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.

This part is already done in the initial code for lab3.

Code will be separated into 3 layers:

* Application engine app.py. It gets user requests and delegates them to appropriate functions to handle; responds with JSON formatted data
* Resources layer. It processes requests. It has business logic of the application, communicates with data layer, formats responses according to the application requirements
* Models layer. Presents data storage API; allows to find /retrieve /save /delete /update data

# Stage2.

1. Implement data storage module models.quote\_module.

Given the following data

**currency = ['USD', 'GBP', 'EUR', 'AUD','JPY']**

list of all supported currencies

**currency\_keys = ['USD\_EUR','USD\_GBP', 'USD\_AUD','USD\_JPY']**

list of all supported combinations between USD and other currencies

**convertion\_rates = [0.85, 0.74, 1.33, 110.71]**

contains rates between USD and other currencies in the same order as in **currency\_keys** list

You will need to populate dictionary **quote\_dict**..

Example:

**quote\_dict[‘USD\_EUR’] = 0.85**

You will end up with 25 keys in the dictionary: 5 for each currency

You can validate your code by running **models.quote\_module.py** from command line:

**python currency-exchange/models/quote\_module.py**

1. Implement 3 following methods to data access and modifications in your storage module:

**def find\_by\_currency\_key(currency\_key):**

**pass**

**def save(currency\_key, convertion\_rate):**

**pass**

**def find\_all():**

**pass**

1. Check the test module **tests/currency\_exchange\_tests**

**class TestQuoteModelMethods** defines the new tests.

run this group of tests and add 3 more tests to it:

**python -m unittest tests.currency\_exchange\_tests.** **TestQuoteModelMethods**

1. Run your application with the use cases below logging to a log file:

**python3 app.py 2>log**

**python3 app.py 2>>log appends to an existing file**

***Test use cases:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **action** | **currency\_from** | **currency\_to** | **amount** | **conversion\_rate** |
| 1 | Q | USD | EUR | ---- | ---- |
| 2 | S | USD | GBP | ---- | 0.74 |
| 3 | D | ---- | ---- | ---- | ---- |
| 4 | X | GBP | AUD | 12.12 | ---- |
| 5 | Q | AUD | EUR | ---- | ---- |
| 6 | X | AUD | GPB | 109.09 | ---- |
| 7 | S | EUR | AUD | ---- | 1.59 |
| 8 | D | ---- | ---- | ---- | ---- |
| 9 | S | JPY | AUD | ---- | 0.012 |
| 10 | Q | JPY |  | ---- | ---- |
| 11 | X | USD | AUD | 237 | ---- |
| 12 | D | ---- | ---- | ---- | ---- |
| 13 | E | ---- | ---- | ---- | ---- |

***Code structure for lab3***

The project code contains several files with the new function prototypes for Lab3 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**

**├── 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.