Solution Design Document for ‘Exchange Rate’ application

Version: 1.00

Updated: 09/04/2024

Author: Bipin Chaskar

# Introduction

The purpose of this document is to describe the IT solution design for the exchange rate application.

The purpose of the solution design is to:

* Brief about the end-to-end IT solution
* Provide approach, architecture, best practices
* Provide all views of the solution required for design, build, testing and implementation

# Requirement Details

* Connect to any exchange rates API (Sample - [Exchange Rates API](https://exchangeratesapi.io/)) to get exchange rates of Australia to New Zealand for the past 30 days into json output format
* Pre process the data to manage any expected issues
* Perform some data analysis
  + Find the best and worst exchange rates for that time period
  + Calculate the average exchange rate for the month
* Feel free to add any additional features or optimizations that you think are relevant and show off your skills.

# Assumptions and Prerequisites

* Make use of open-source python packages
* Store json results for reference at the same location
* Current date will be used for date duration

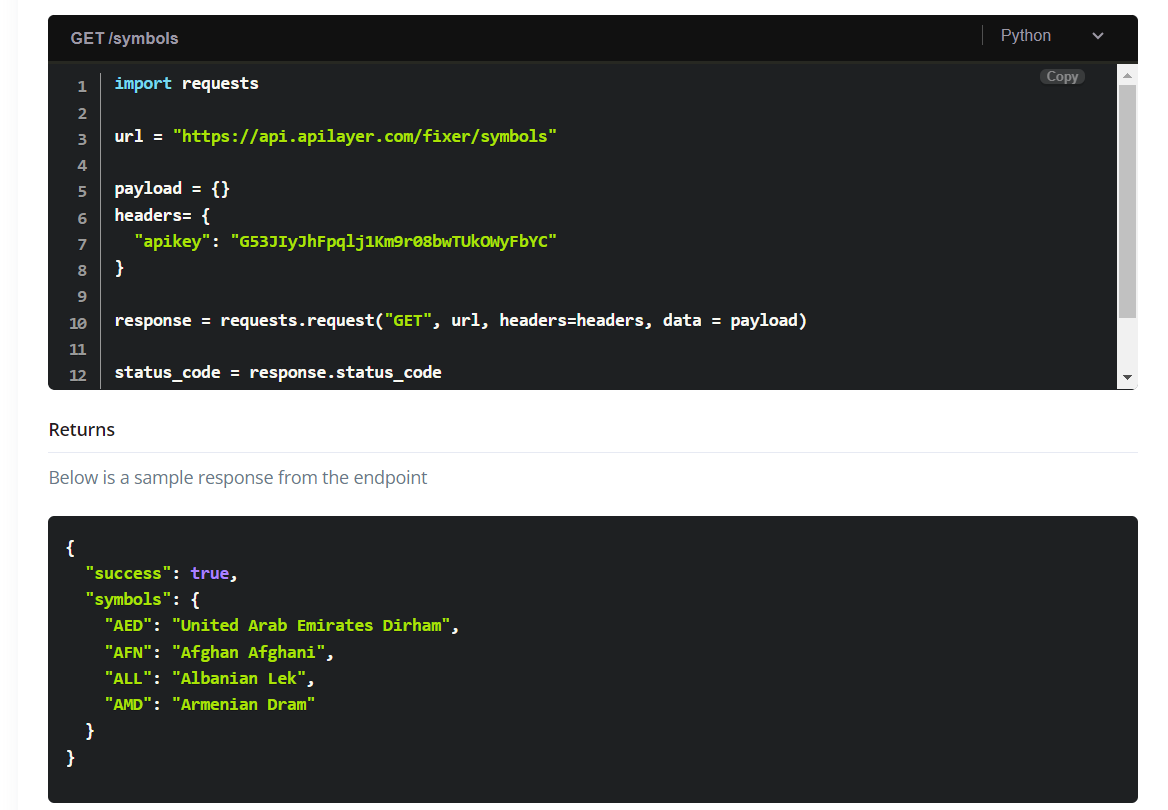
# Solution overview



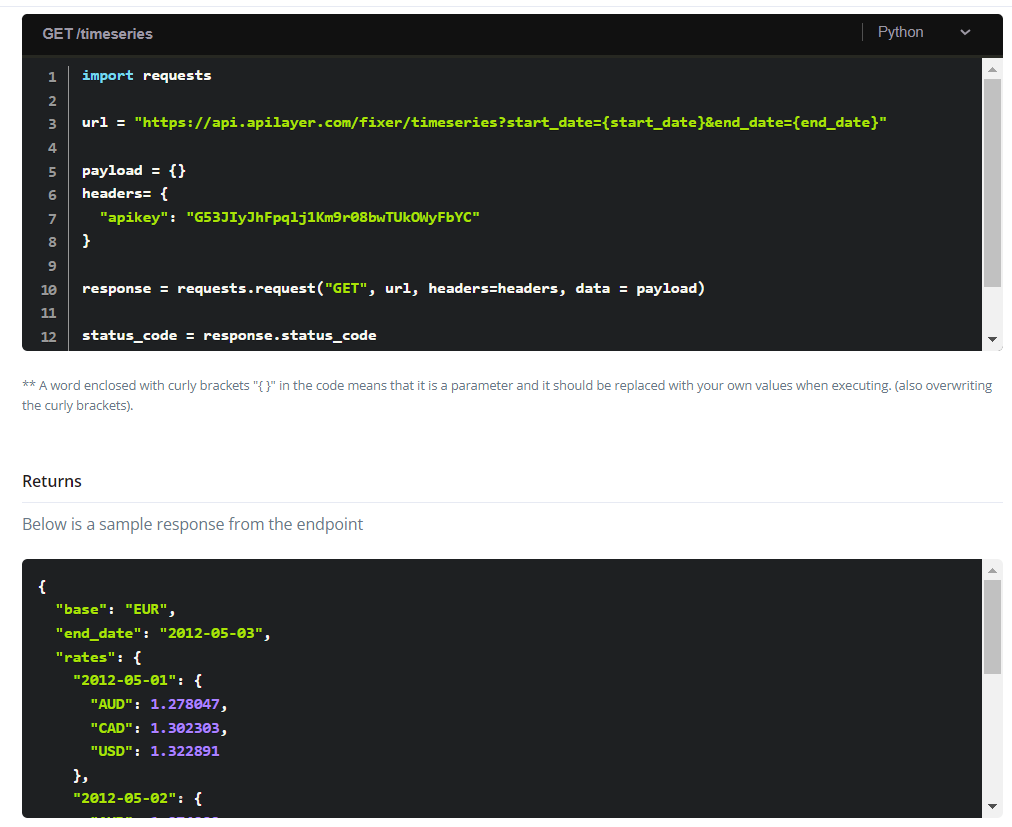
## 4.1 Exchange rate API:

The currency APIs have various endpoints available to use within the application. Each endpoint requires certain arguments to return the desired result. Refer [API Info](https://apilayer.com/marketplace/fixer-api?utm_source=apilayermarketplace&utm_medium=featured) for more details.

1. Subscribe to the Fixer API and generate API key
2. Find out the endpoint to use for this use case
   1. Symbols: This endpoint returns all available currencies.



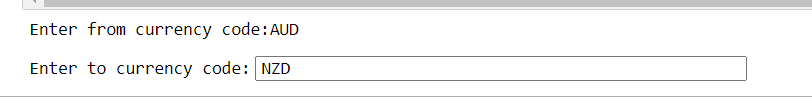
* 1. Timeseries: This endpoint lets you query the API for daily historical rates between two dates



## 4.2 User inputs:

Prompt user to enter following inputs

1. From currency. This will also be used as base currency.
2. To currency.



Get the latest list of currency code based on user inputs and stored in the list data type.

Perform input validation using following rules

1. Currency code is only alphabets
2. Currency code is three letters
3. Currency code is valid as per API supported currency code list

## 4.2 Extract data:

Construct the URL by merging base URL with user inputs. Use API key and URL to connect with API and return JSON contents. If the URL is invalid then capture the error in the exception log.

Save content is JSON format.

{

"success": true,

"timeseries": true,

"start\_date": "2024-03-09",

"end\_date": "2024-04-08",

"base": "AUD",

"rates": {

"2024-03-09": {

"AUD": 1,

"NZD": 1.079

}

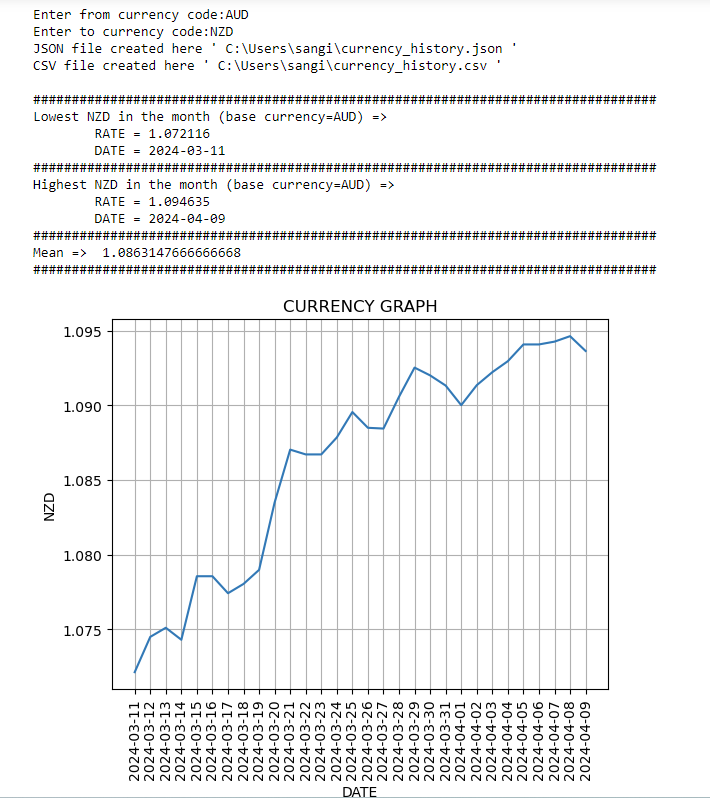
}

## 4.4 Pre-process:

The JSON key value pair needs to be parsed so that mathematical operations can be performed against data.In this case the CSV writer library is used to pick only required JSON key value pairs and parse them into comma separated columns. Headers are added to the file to use them in data analysis.

## 4.5 Results:

Finally, the results for different data analysis queries are printed on the terminal.



# 5. Best practices

* Use Built-In Functions and Libraries. Built-in features are already optimized for performance, so use them whenever possible.
* Formatting & Syntax Best Practices.
* Reusability in the code
* Documentation in the code
* Error and exception handling in the code
* Avoid Excess/Unnecessary Data Structures.

# 6. Challenges

Some of the challenges faced during the development of this project include:

* Integrating external APIs and handling API responses. APIs provide a limited number of hits for free access.
* Implementing proper error handling for API calls.
* Parsing JSON into CSV/Text format to store values in columnar format.

# 7. Test case:

Refer to the excel sheet.