Table of Contents

1. Introduction to Crypto Portfolio Command line Programme

```
SAJEEWA@Sajeewa-Nitro-i7 MINGW64 ~/Desktop/propine/crypto_portfolio (dif-approach)
$ npm start

> crypto_portfolio@1.0.0 start
> node app.js

Welcome abroad !
This is Node.js CLI program to check portfolio value: You can go for following options
1. Latest Portfolio per token in USD
2. Latest Portfolio for a specific token in USD
3. Portfolio per token in USD on a specific date
4. Portfolio of a specific token in USD on a specific date
Refer to the table below for commands
```

Option	Commands	Arguments	Example
1	latest		latest
2	latest	--token=<token>	latest --token=BTC
3	date	--date=<date>	date --date=2017/03/05
4	date	--date=<YYYY/MM/DD> --token=<token>	date --date=2017/03/05 --token=ETH

```
Loading csv ...
initial Loading time: 50.114s
CSV file read and processed !
Please Enter command : latest

  Latest Balance

   ETH: 1588911151.4 $
  BTC: 33271883836.4 $
  XRP: 404512.5 $

Please Enter command : latest --token=ETH

  Latest ETH Balance

   1588829998.0 $

Please Enter command : date --date=2017/04/05

  2017/04/05 : Total Balance

   ETH: 38204744.3 $
  BTC: 1282714332.8 $
  XRP: 30419.2 $

Please Enter command : date --date=2017/04/05 --token=XRP

  2017/04/05 : XRP Balance

   30419.2 $
```

This is an attempt to create a program that manipulates a CSV file containing daily user transactions.

The program should provide four types of results:

- The latest portfolio value per token in USD (when no parameters are given)
- The latest portfolio value for a specific token in USD (when a token is given)
- The portfolio value per token in USD on a specific date (when a date is given)
- The portfolio value of a specific token in USD on a specific date (when a date and token are given)

The data includes timestamps, transaction types (DEPOSIT or WITHDRAW), tokens (BTC, ETH, XRP), and token amounts.

2. How to use

- Download the repo and Install dependencies:

  npm install

  Important:

  Make sure transactions.csv file is saved in **crypto portfolio** folder.

- Run the application by using following command and then follow the instructions thereafter

  npm start

Option	Commands	Arguments	Example
1	latest		latest
2	latest	--token=<token>	latest --token=BTC
3	date	--date=<date>	date --date=2017/03/05
4	date	--date=<YYYY/MM/DD> --token=<token>	date --date=2017/03/05 --token=ETH

Note:

Application will take some initial boot time depending on your computer performance and size of the csv file. Thereafter it will continue to run smoothly

3. Language Stack and packages used

a. Node.js:

Node.js is an open-source server environment that allows developers to run JavaScript on the server-side. It provides a platform for building fast, scalable, and highly performant network applications using an event-driven, non-blocking I/O model.

b. Axios package:

Axios is a popular Promise-based HTTP client for JavaScript that can be used in both browser and Node.js environments. It provides an easy-to-use API for sending HTTP requests and handling responses, supporting features such as interceptors, request cancellation, and automatic JSON parsing.

c. Chalk@4.1.2 package:

Chalk is a Node.js package that provides a simple way to add color and styling to the console output. It is used to customize the look and feel of the command-line interface (CLI) by adding color to text, changing the background color, and applying other formatting options.

d. Yargs package:

Yargs is a Node.js package that helps to build interactive command-line tools, by providing a simple and intuitive way to define commands, arguments, and options. It makes it easy to create powerful CLI applications by abstracting away much of the complexity of handling command-line input.

e. Boxen@5.1.2 package:

Boxen is a Node.js package that allows developers to create nicely formatted boxes in the console. It provides a simple way to create visually appealing terminal messages, complete with custom borders, colors, and other styling options.

f. Csv-parse package:

Csv-parse is a Node.js package that provides a way to parse CSV files and convert them into a JavaScript object. It can handle complex CSV data with support for features like custom delimiters, line breaks, headers, and more.

g. Console-table-printer package:

Console-table-printer is a Node.js package that allows developers to create tables in the console output. It provides an easy-to-use API for generating tables with customizable headers, footers, and cell styles.

h. Nodemon package:

Nodemon is a Node.js package that helps to develop Node.js applications by monitoring for any changes in the code and automatically restarting the server. It makes the development process more efficient by reducing the need to manually restart the server after each change.

4. Various design decisions

I was able to solve the problem using two different approaches.

a. First Approach

In this approach, the application reads a CSV file and processes data for each of the four requests. The yargs package is used to ask for commands. Since the data was fairly large, a simple read of data was not possible. Therefore, the data was read as a stream line by line and processed as data come and assigned to an object. To handle this data stream easily, the csv-parse npm package was used. However, for each command that the user requested, the application needed to go through all the data in the CSV file, which is nearly 1GB in size. As node.js normally runs on a single thread, we have to wait for this single thread to read and process all the data. In this approach, it would take around 60s to give output for each request.

To overcome this issue, **worker threads** were used to read the CSV file from multiple points in parallel using extra threads allocated for node.js in libuv thread pool for asynchronous tasks like reading a file. There were two options to divide the CSV file between each thread:

I. Simply measure the file size of csv file and divide by number of threads : but some data leaks happens in this approach.so I have decide to not to go with it
II. Assign each threads to read predefined number of records.
Example:
here I have optimize the application for reading 30 Million of csv records by dividing 30 million records between each thread, also I have kept open last worker thread's end limit undefined , so I can use this application to read any number of records.

Here data processed fine, so I have decide to go with this approach.

By using threads to execute the programme I have mange to reduce execution time by **50%** (30s)But I found following drawbacks in above approach.

the program had to read the CSV file again and again for each request. This meant that each request would take around 30s to output the result. Therefore, the second approach was chosen to overcome this drawback.

b. Second Approach

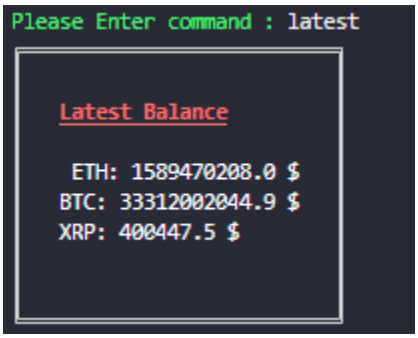
To avoid the drawback of the first approach I have decide to go with following approach to the problem.

I. In the first execution of the code, programme is reading the data while processed these data in a way, it will save the cumulative balances up to date in an object that uses relevant date as the key.

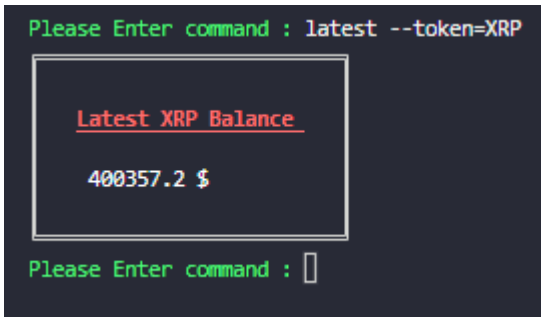
```
cumulativeBalances = {
  '7/24/2017': { ETH: 231, BTC: 314, XRP: 275 },
  '7/23/2017': { ETH: 177, BTC: 245, XRP: 217 },
  '7/22/2017': { ETH: 140, BTC: 190, XRP: 183 },
  '7/21/2017': { ETH: 80, BTC: 121, XRP: 129 },
  '7/4/2017': { ETH: 46, BTC: 67, XRP: 60 },
};
```

II. So then I simply have to go through that object to find the needed data for each option like below.

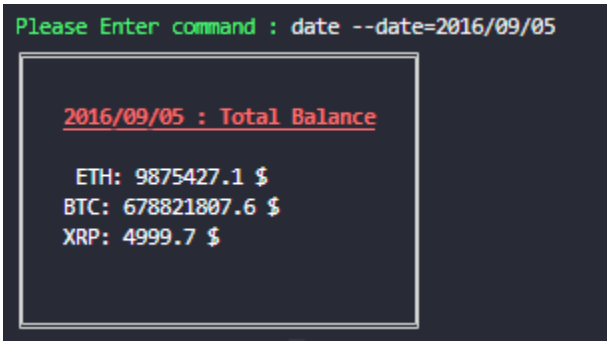
a) To return the latest portfolio value per token in USD, the program will search for the latest date key in the object and retrieve the corresponding token values. Then, it will request the latest USD rates for each token from the cryptocompare API by using the inner object keys. The program will multiply the token values by their respective USD rates to obtain the portfolio value per token in USD.



b) To return the latest portfolio value for a specific token in USD, the program will search for the latest date key in the object and retrieve the token balance for the specified token. It will then request the latest USD rate for that token from the cryptocompare API and multiply the token balance by the USD rate to obtain the portfolio value of the token in USD.

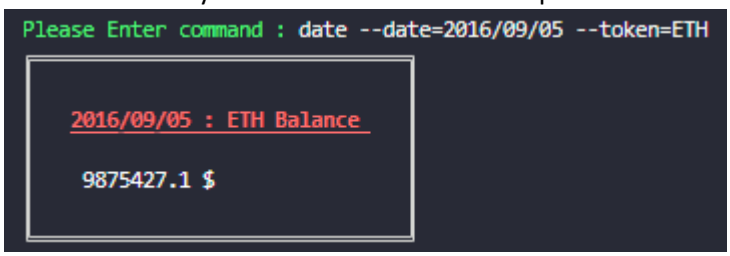


c) To return the portfolio value per token in USD on a specific date, the program will search for the date key in the object and retrieve the corresponding token values. If the date key is not available, the program will use the latest available date before the specified date. The program will then request the historical USD rates for each token from the cryptocompare API and multiply the token values by their respective USD rates to obtain the portfolio value per token in USD on the specified date.



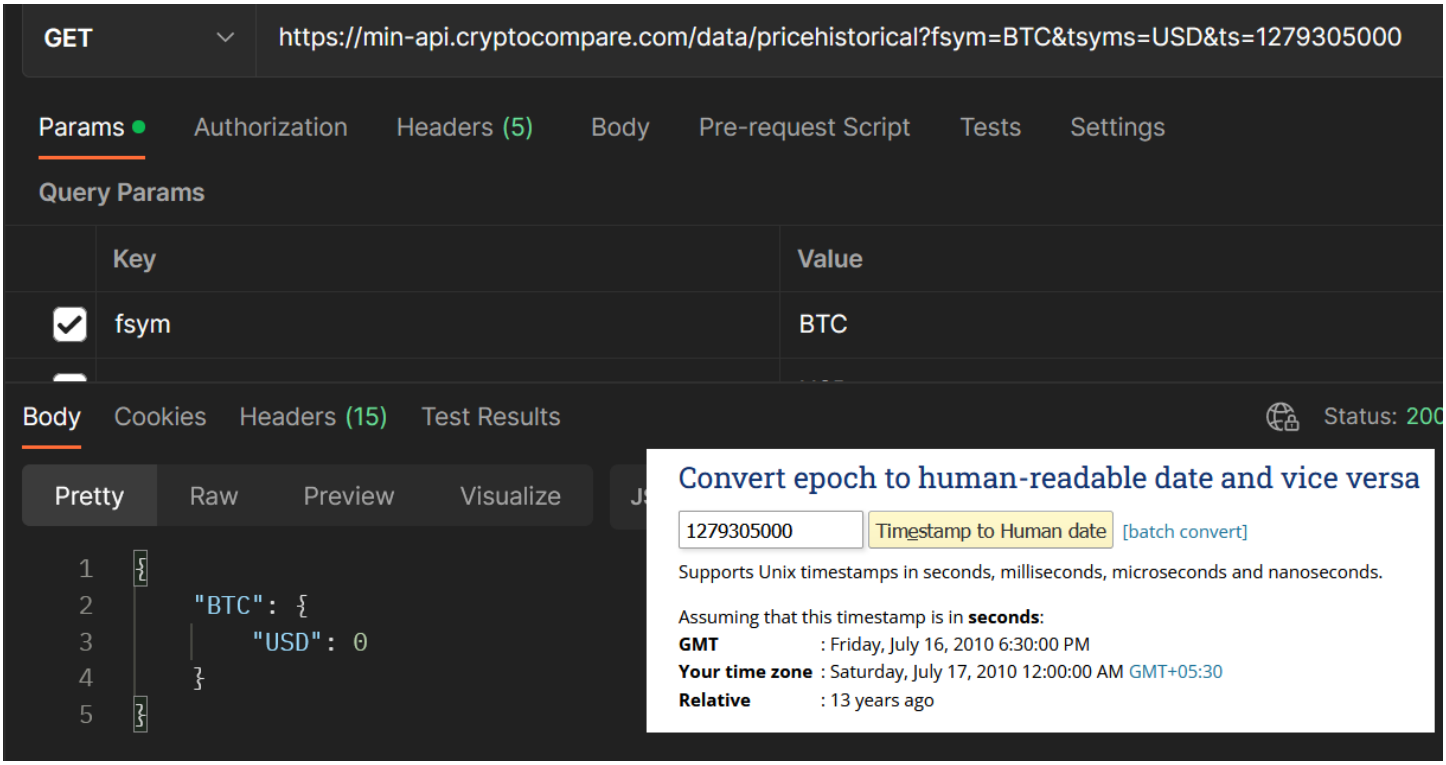
d) To return the portfolio value of a specific token in USD on a specific date, the program will search for the date key in the object and retrieve the token balance for the specified token. If the date key is not

available, the program will use the latest available date before the specified date. The program will then request the historical USD rate for the specified token from the cryptocompare API and multiply the token balance by the USD rate to obtain the portfolio value of the token in USD on the specified date.



- One of the major problem:

While getting historic data is in crypto compare API data is not available beyond Sunday, July 18, 2010 12:00:00 AM GMT+05:30.beyond that date USD rates of crypto currencies are zero. (Of course since crypto not available beyond the particular time)



5. Error Handling

In this programme there are major events that causes errors.

a. When reading csv file

There are several potential errors that can occur when reading a CSV file in a Node.js program

- File not found: If the CSV file does not exist or is not in the correct location, the program will generate a "File not found" error. This I have handled.

```
SAJEEWA@Sajeewa-Nitro-i7 MINGW64 ~/Desktop/propine/crypto_portfolio (dif-approach)
$ node app.js
```

Option	Command	Arguments	Example
1	latest		latest
2	latest	--token=<token>	latest --token=BTC
3	date	--date=<date>	date --date=2017/03/05
4	date	--date=<YYYY/MM/DD> --token=<token>	date --date=2017/03/05 --token=ETH

```
application can't find the csv file

Please make sure the csv file with the records, transactions.csv file available in
C:\Users\SAJEEWA\Desktop\propine\crypto_portfolio path and try again !
```

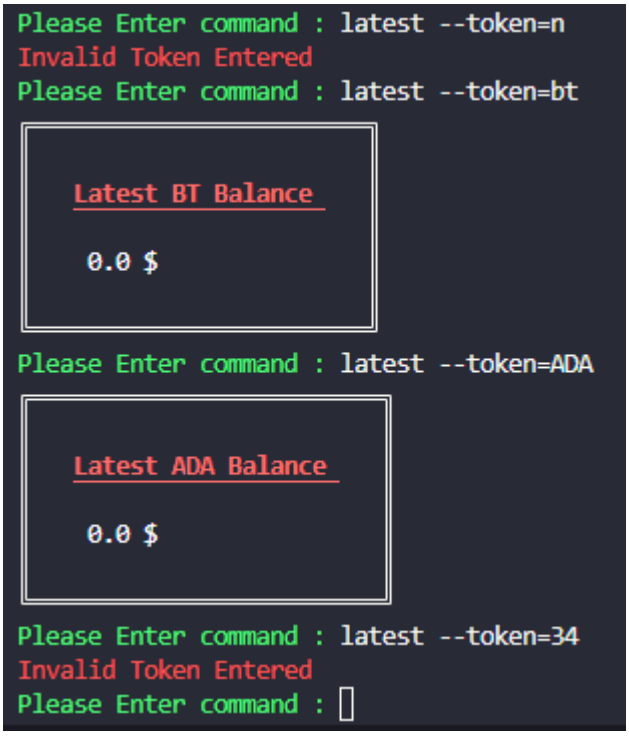
- File permissions, CSV format, Encoding issues, Memory issues and Disk I/O errors commonly handled

b. When interact with the user for commands, user may enter wrong commands or arguments or both

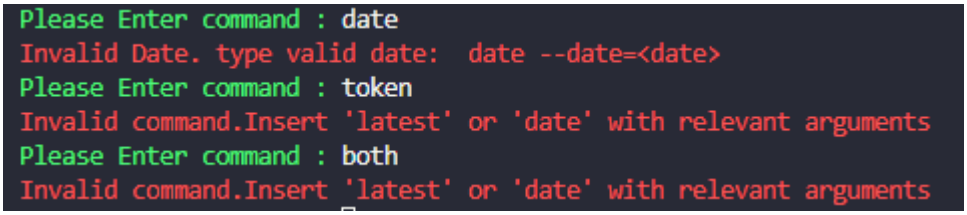
- I have added validator to validate dates from the user

```
Please Enter command : date --date=2015/02
Invalid Date. type valid date:  date --date=<date>
Please Enter command : date --date=2015/02/30
Invalid Date. type valid date:  date --date=<date>
Please Enter command : date --date=svdf
Invalid Date. type valid date:  date --date=<date>
Please Enter command :
```

- Also checking with crypto compare api for valid token

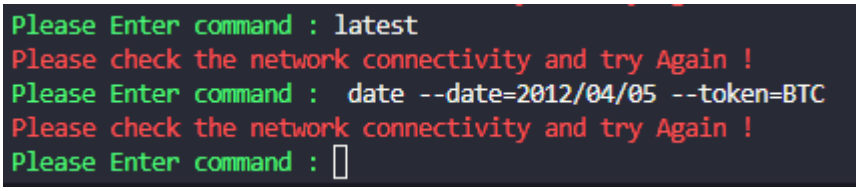


- Also, Error handling added for different user commands other than the defined ones.



c. When requesting data from crypto compare using axios

Following results are taken while airplane mode on pc



6. Glossary

a. Worker threads in Node js

Worker threads is a built-in module in Node.js that provides a way to run JavaScript code in separate threads, enabling developers to execute CPU-intensive tasks without blocking the event loop. With worker threads, Node.js applications can leverage the full power of modern multi-core CPUs by creating and managing multiple threads, each with its own event loop and shared memory space. This can result in significantly improved performance and reduced response times for applications that require heavy processing or large amounts of data. Worker threads also supports message passing between threads, allowing developers to exchange data and coordinate work between threads in a safe and efficient manner.

b. Libuv

Libuv is a multi-platform support library that provides asynchronous I/O operations, networking, and event-loop functionality for Node.js. It abstracts the underlying system differences between Windows, macOS, and Linux, providing a consistent API for network and file system operations.

Node.js utilizes libuv to handle non-blocking I/O operations and to manage events in the event loop. It provides an event-driven, non-blocking I/O model that allows Node.js to handle high concurrency with a minimal amount of overhead.

By default, libuv uses a thread pool with four threads to perform I/O operations. This thread pool size can be increased by setting the UV_THREADPOOL_SIZE environment variable to a value greater than four.