INTRODUCTION:

• Our project aims to quantify investor performance and compile it in a mySQL relational DB made up of table that feature trader name, trading score (generated by us after they play our simulator), and a gratuitous amount of crypto awarded depending on the score they achieve. The simulator will determine the proportion of digital currency for the investor to retain beyond the initial stage of the platform. This is to incentivize positive investor conduct without compromising trader anonymity (blockchain advantage), waiting for regulation or compromising accessibility to trading platforms.

BACKGROUND

Our software is designed to be an add-on to existing trading platforms to increase the efficiency of the financial market as well as the experience of traders themselves. Using a digital device to buy and sell shares has increased participation in the stock market. This can bring several advantages: more people actively borrowing or lending generated increased financial literacy among the population which in turn can lead to greater productivity. Similarly, higher financial activity can incur higher investment which improves the flow of money. On the other hand, it can also diminish market efficiency and since the secondary market can be greatly influenced by speculation, limiting noisy trading can have a significant impact.

<u>Mechanism:</u>

New traders to existing platforms can sign up via our add-on. The add-on will require them to pass through a simulator before they can inscribe onto the database. This simulator will provide them with a score and a gratuitous amount of crypto currency depending on their performance.

<u>Motivation</u>

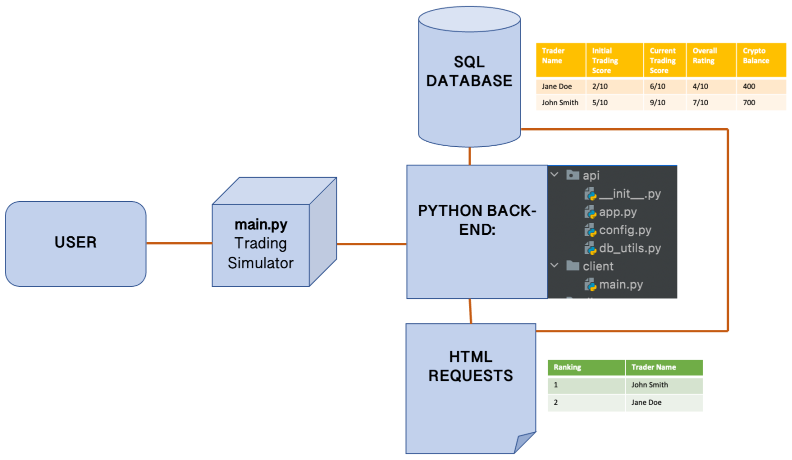
Every investor's aim is to out-perform the market, the sole incentive of trading is to generate profits. The project is underpinned by the assumption that practices such as insider trading or money laundering, yield negative utility to every investor so as to exclude it from this report's analysis. Although they are real-life examples of market failure which this project's concept could address, investors who use illegal conduct don't fit the rational investor definition as the probability and consequence result in a net negative payoff despite it possibly generating higher profits than compliant trading.

<u>Mechanism</u>

Trading platforms incorporate a level of stewardship over their users by verifying accounts through the back-end to varying degrees of security; some platforms will just require a UK bank account and email address while other specialist ones would even require legal forms of ID and an identity check. While these address the issues of maintaining accountability to deter misconduct, no measures are in place to motivate informed investment decisions. Incorporating incentives into an already standard practice will encourage initial compliance that is anticipated to lead to sustained improvement, largely because it requires changed behaviour rather than blind obedience. A trial period on investment platforms is already mainstream and many platforms mandate using game/demo/provisional currency first before depositing real cash. This solely limits the benefit of informed trading to the investor themselves; responsible trading reduces their risk. Our software aims to share the benefit by ensuring informed trading takes place beyond the probation period.

Further development will be made in unit testing to ensure several criteria: that suspected speculative trades will not be inputted, that the calculated ratio/proportion of retained profits cannot be bypassed by the investor and is solely DB derived, that users cannot repeat the probation period as a way to obtain more, that users must reinvest retained profits, they cannot simply just complete the probation period and withdraw.

SPECIFICATIONS AND DESIGN



Trader Name	Initial Trading Score	Current Trading Score	Overall Rating	Crypto Balance
Jane Doe	2/10	6/10	4/10	400
John Smith	5/10	9/10	7/10	700

Ranking	Trader Name
1	John Smith
2	Jane Doe

1. Client interface

The client interface will welcome new users to a given trading platform (our software acts as an add-on to existing platforms). The user will be able to create an account or check existing trader stats. The following functions are offered to the user: display best, worst and all traders. Get a trader row by searching their ID and add a new traser. If the client wishes to make their own account, they will first be asked to partake in a trading simulation with mock digital currency. The trading simulator will generate a score for them which they can input along with their name when completing their account.The add_trader() function will use a PUT request to add a new trader to the website. Their score will be used to determine how much gratuitous crypto currency they start trading with on the platform. The better their score, the more digital currency they are offered.

2. Database Utils

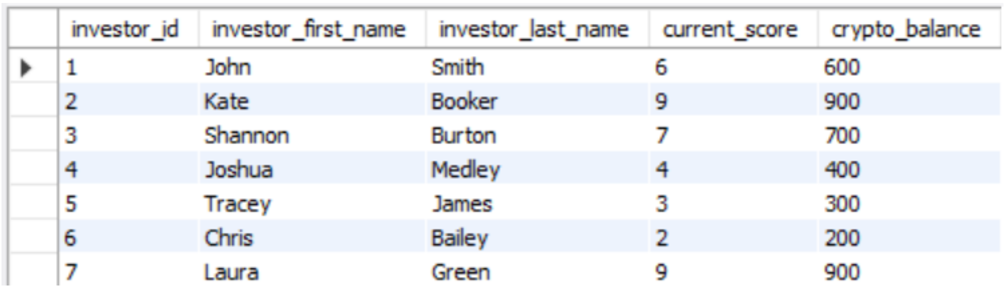
Contains logic and methods which correspond to the functions contained on the client side. These functions connect the database to the python backend. In each function is stored a string with formatting options for the data passed to the function. These functions will do one of 2 things in the database: create or retrieve rows.

3. App.py

Contains the functions and HTML end points for executing the PUT and GET HTML functions.

• Design and architecture

Database Design



	investor_id	investor_first_name	investor_last_name	current_score	crypto_balance
▶	1	John	Smith	6	600
	2	Kate	Booker	9	900
	3	Shannon	Burton	7	700
	4	Joshua	Medley	4	400
	5	Tracey	James	3	300
	6	Chris	Bailey	2	200
	7	Laura	Green	9	900

The database we created is named nano. It consists of one table named investors_info. This table is shown in the diagram above. It is filled with example data. Each row represents a new investor that has registered with the app and the information obtained about them. The columns consist of investor_id, investor_first_name, investor_last_name, current_score and crypto_balance. Investor_id is a unique identifier for each investor. This has been used as the primary key. The next two columns are first and last name. These names have been obtained through being inputted on the simulator. The current score gives the investors score that has been obtained through the trading simulation. The higher the score the better the trader. The crypto_balance highlights the amount of crypto currency made through the simulation. This should correlate with the current_score.

Simulator Design

Modules used: requests, time, random, itertools and math

OOP has been used in storing the methods associated with money to be invested. The class investment stores the amount and currency type as input values. It also generates a trading score and free crypto balance depending on how well the player performs. Initialisation of the class looks like this:

```
class investment():

    def __init__(self, currency, amount):

            self.currency = currency

            self.amount = amount

            self.score = 0

            self.crypto = 0
```

There are 2 functions stored in the class (fluctuation() and gamble()). The former initializes a fluctuation in the current crypto balance based on a random 2 decimal place float number. The latter will offer the player a chance to gamble their digital stock. If their balance is 3 numbers or less the gamble function will scramble the numbers and pick a random permutation. If the balance is more than 3 numbers the gamble function will offer a 1 in 500 chance of increasing the balance by a factor of 10. There are 2 options in the gamble function as larger numbers require too much processing time to form permutations. The math module was required for creation of both functions and itertools was required for the creation of the gamble function (permutations). Recursion has been used in the gamble function within the first if statement. The recursion function calculates the number of permutations that will be generated from the balance. This is important as in order to pick a random permutation we first need to know how many items will be in the iteration object (and len() is not polymorphic to iteration objects). The num_permutations() function is defined outside of the class and called within the generator() function. It works by using classic factorial recursion logic.

Both the score and free crypto currency balance are determined by 1 function in the investment class. This function generates a score first by taking the logarithm base 2 of the balance. The free crypto given to the user is then the squared result of this logarithmic value. This is an arbitrary equation made with the purpose of generating 2 numbers that can be used to grade and reward traders.

Logic for the score and crypto generation:

```
def free_crypto(self):

    self.score = round(math.log(self.amount, 2), 2)

    self.crypto = round((self.score **2), 2)
```

The switch_currency() function is one of the larger functions in the simulator. It requires an API which makes calls to COINapi.io - a free API that stores digital currency exchange rates. The methods contained within the function have been implemented with the max number of API calls per minute (5) in mind. Therefore I have chosen 4 currencies that can be switched between, to allow for each exchange rate to be displayed (4 calls) while saving 1 API call for the exchange once the user has input which currency they would like to switch to. In order to protect the simulator from exiting due to the switch_currency() function being called multiple times if needed, I have designed a decorator function. This is a timer function which will allow the switch_currency() function to execute but make the user wait 60 seconds before the program will continue. I have added a return to the inner_wrapper() as well as returning inner_wrapper so that the product of the switch_currency() function is delivered to the class investment() smoothly.

Running the simulator in main.py

I have initialized the main.py file to use the simulator in the following order: welcoming the new trader and asking for their details, establishing starting amount of crypto and the type of currency they will play with, offering to switch currency, fluctuating the balance, offering to gamble, final fluctuation and score generation.

The investment() class was very useful for fast manipulation of the balance in the main.py file. I imported the collections module into the main.py file so that a Named Tuple could be used to neatly store the output of the simulator for later passing to the SQL database.

IMPLEMENTATION AND EXECUTION

- Development approach and team member roles
- Tools and libraries

- Implementation process (achievements, challenges, decision to change something)
- Agile development (did team use any agile elements like iterative approach, refactoring, code reviews)
- Implementation challenges

Development approach

We decided to use an agile style approach to develop our app. This involved developing our app in smaller increments to ensure it is constantly evaluated and ensuring that we responded to challenges rather than following a strict plan.

We first planned our project. This involved ensuring all team members knew the aims of our project, why we were conducting the project and planned how the aims would be met. In this time, we determined each team member's strengths and therefore designated roles for each team member. These are shown below. Despite these given roles we were all able to help each other if needed, by offering support on more challenging aspects of the project. We also clearly identified the main features of our project to ensure all team members were working towards the same goal.

When implementing the project, we had regular meetings (every 3-5 days) to check up on the progress of everyone’s work. In these meetings each team member would discuss the work they are currently working on, the progress they have made and any challenges they have faced. This allowed the team to discuss the progress and help solve any challenges. When facing these challenges there would be times that the project needed to be adapted. This would be discussed as a team to ensure all team members understood the adaptation and could still progress to their end goal. Each team member would then tell the team what they would aim to do before the next meeting.

In terms of the timeline of the project we first decided to design and code the mySQL database, create the structure of the API and begin designing and coding the trading simulation. These steps all took place independently. The API is then designed and coded with the features described in the specification and design section. Once individual functions are complete these were tested in order to ensure our code works efficiently. The whole programme will be tested at the end of the project to ensure different scenarios work effectively.

Team Member Roles

In the initial meeting each team member was given a specific role to fulfill in this project. These roles were:
Rebecca - Design the database storing all information on the investors and code any SQL syntax needed in the database API.
Emily - Design and execute the trading simulation, build empty API structure + write API functions
Iman - Design and code the API which will include writing code for the application and the db utils.
Amanda- Assisted with coding the API and had the most prior knowledge on trading/ investing therefore was able to help others understand the background to the project.
Promise- Define the testing strategy and complete all unit testing to ensure the code is working

efficiently.

Tools and library
To create the database, we used mySQL. Mysql.connector allowed us to connect the MySQL server to our API in Python. To create our API we used Flask and the jsonify module to return JSON data. Listed below are the libraries we used:

- Requests - to send HTTP requests.
- Json - to allow us to work with JSON data.
- Collections - to create a tuple with subclasses so we can easily access data.
- Math
- Random
- Itertools
- time

Implementation process (achievements, challenges, decision to change something)

Simulator implementation: successfully called API to get live exchange rates. Use of a timer decorator in order to safeguard against API requests reaching max and program exiting. Use of recursive function to calculate the possible number of permutations in the gamble function. Challenges included changing the API provider last minute due to the original provider moving some of their services to their premium subscription instead of their free users. This resulted in the decision to change to an alternate provider.

Achievements
We were successful in building the API and the trading simulation which was then tested.

Challenges
When building the API, it was initially difficult as we did not have much practice. However, we were able to complete it by rewatching Lesson-20. We found issues when attempting to implement the _mapped_values function (db_utils.py) into the function that allows users to select an individual user to view in the HTML. We then managed to overcome this by using dictionary comprehension. However, when getting the data from the api into the client-side, the values for a specific trader gave several errors. Perhaps this was due to the same error in the db_utils file. We then made a list of the values from the dictionary obtained from the api and used the indexes of the list to then format it for the client side.
Another challenge faced was testing the simulator. Whilst it appeared to work well when running on Python we tried to conduct unit tests on the individual class methods to ensure they all worked as expected. It was difficult to test these methods as their output was print statements rather than returns and we were unable to retrieve some variables such as fluctuations. This made it difficult to form an accurate test. These issues paired with the time constraint made us unable to test the simulation through unittest.

Changes
We initially planned to add a rating function into our program where users could rate each other based on trades made with one another. However, we realized this may cause issues and bias as users could give their friends or people they like high ratings and low ratings for others.
As this program is not a stand alone, there was not a method for us to allow users to withdraw their

earnings as this would have to be done with the main trading app that works with our add-on.

Agile development

Throughout the project, we had regular meetings where we could discuss progress, what each person intends to do next and whether anyone needs help on their tasks. At each meeting we also held code review discussions where members can evaluate code logic and discuss the best approach to a problem.

Implementation challenges

To obtain exchange rates between currencies we implemented the Alpha Vantage API, however, due to changes by the API provider, the exchange rates were behind a paywall as of 14/08/2022. The COINapi was then used for exchange rates.

TESTING AND EVALUATION

• Testing strategy

For the testing strategy, we used the analytical based testing strategy. The endpoints and functions are tested to ensure that they meet functioning requirements.

• Functional and user testing

We used Functional testing(unit test) to test that the routes/endpoints are properly working. The unitest starts by defining a class and taking in some defined functions. The url of all routes in the API is passed to the function to check their response. If the endpoints are responsive, then the test will pass, else it will return an error.

• System limitations

A major limitation is that the unittest wasn't able to test how good the system can perform, its usability and scalability too.

CONCLUSION

To allow investors to make more informed investment decisions we have created an app/ ad on app which will allow an investor's performance to be quantified. An investor's performance will be quantified by taking part in an initial trading simulation which replicates market fluctuations and allows a user to gamble their money or switch currency. At the end of this simulation a score is given based on your investment decisions during the trading simulation. The higher the investor score the more reputable the trader. The investor score is stored in a database along with the investor id, investor first name and last name, investor crypto balance and currency. The investor crypto balance and currency are also gained from the trading simulation. The app then allows a user to access details in the database such as individual investors stats, display all the investors scores in order of the score and show the best and worst investors. This will all be done without compromising an investor's identity (as only an investor's id will be shown). The information gained from this app should allow investors to make more informed decisions when investing. In the future this app could be developed to allow previous trades to be inputted by an investor to improve the scores accuracy.