Cryptica Social Media Analysis Application

NEA Document

Arthur Robertson

Centre Number: Redacted

Candidate Number: Redacted

Contents

ANALYSIS

DESCRIPTION OF PROJECT

For my project I decided that I want to build a modern secure web application that follows good security practices. I have decided to find a client to develop an application for that will allow me to develop an authentication system alongside their desired functionality. I have some experience in searching for website vulnerabilities, which I will be putting to use when creating and testing my application.

BACKGROUND ANALYSIS

My client is an IT professional who works for a company maintaining their IT systems. My client keeps his finances in order, and puts part of his salary into a variety of investments, such as stocks and shares. My client has a small portion of his investments in Cryptocurrencies, however he would like to expand on his investments and increase it's stake.

Cryptocurrencies are a new asset class that has seen a dramatic rise in popularity over the past year. They are attractive to retail investors as they typically offer high returns with high risk. The cryptocurrency markets consist of hundreds of coins, that each perform differently and have different values and purposes. They tend to be extremely volatile, seeing huge gains and losses in very short periods of time. The market also tends to be influenced by politics and real life events, and are surrounded by controversy.

Figure 1: Bitcoin's price over the last 5 years. Source: Google Finance

My client is a retail investor, and he primarily bases his investment decisions based on his own research. His research primarily consists of analysing news articles, and social media sentiment. My client has noticed that high profile celebrities such as Donald Trump and Elon Musk can have a large impact on the price of assets based on a short social media post. In fact, Elon Musk has previously got in trouble in the past with the SEC for tweeting regarding Tesla's stock. Cryptocurrencies however are a very unregulated market, and Elon Musk has tweeted about them several times with no consequences.

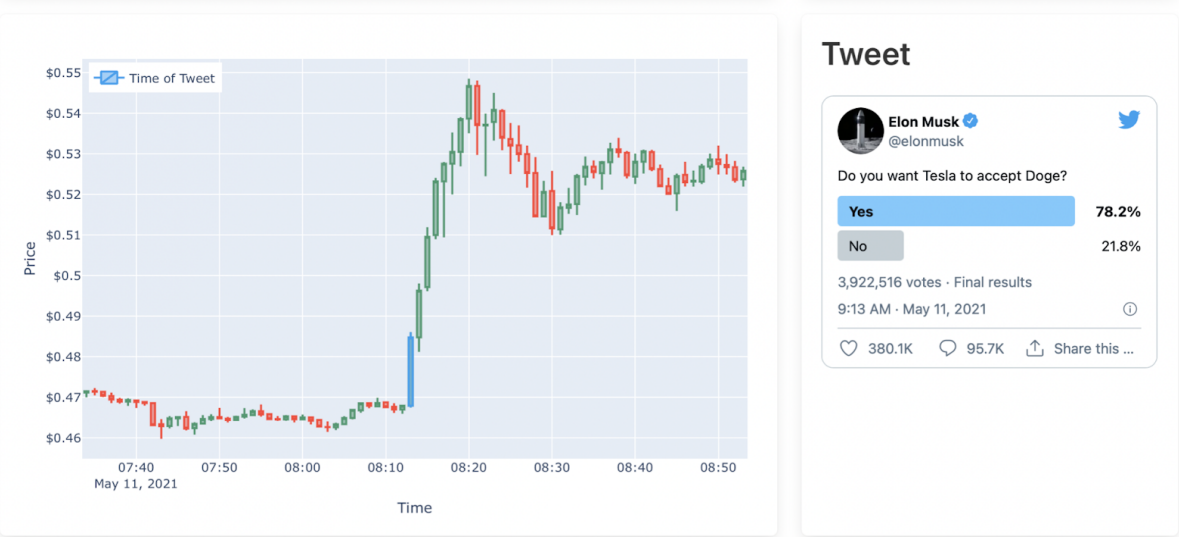

Figure 2: Graph I generated in Python showing the effect of a Tweet on the price of Dogecoin, a cryptocurrency. The blue candlestick on the financial chart shows the time of tweet.

My client wishes for a web application to be able to analyse the effect certain social media posts have on the price of cryptocurrencies, with a program that generates graph such as the one above to show the impact.

With my other clients investments, he keeps informed on current events and news by reading newspapers such as the Financial Times, which offer a range of high quality articles on most things affecting the stock market. Cryptocurrencies however do not typically feature frequently in publications like the Financial Times, meaning that my client has to look elsewhere for news related to cryptocurrencies. He would also like to be able to see a collection of Cryptocurrency related news articles in one place, saving him from searching across multiple publications.

INTERVIEW WITH CLIENT

Me: What functionality are you looking for with this project?

> **Client:** I would like an application that allows me to perform my analysis all in one spot. Currently, my analysis on social media on the cryptocurrency markets is not good, and is

> not helping me make informed trades. I would like a tool that allows me to gain a quick overview on social media opinion on cryptocurrencies.

Me: What do you normally trade on the cryptocurrency markets?

> **Client**: I normally trade a variety of assets, including Bitcoin, Ethereum, and some other smaller “alt coins” such as Dogecoin.. I trade on a platform called ‘Binance’.

Me: How do you base your decisions on what to trade?

> **Client**: I make my trades based on overall sentiment on certain assets. I like to keep an eye out for what assets high profile celebrities are mentioning. I often find that they can have great influence on the price of some cryptocurrencies. I normally search on Twitter for certain users and try and find and analyse how the price of certain things change after they tweet. I like to read articles from news publications as well, especially articles on cryptocurrency though they rarely appear.

Me: Are there any problems with your current trading method?

> **Client**: Yes. Often, by the time I find about certain coins and see them on social media, they have already spiked in price and it is too late for me to invest in them. I also don’t always know who I should pay attention to on twitter, certain users such as Elon Musk tend to be very influential with the markets, but I need a way to verify this. There is a very high quantity of spam on Twitter, and I need to be careful who I pay attention to and who I ignore. In addition as I mentioned before, I like to get my news from news sites and papers. However, I can never seem to find articles about cryptocurrency which is annoying.

Me: What tools are you looking for to help you make informed trading decisions?

> **Client**: I would like a tool where I can analyse a user’s tweets, and check for mentions of cryptocurrencies. I would then like to be able to analyse and work out if said user’s tweets have any influence on the price - this will help me make an informed decision as to whether to follow the Twitter user’s advice on coins.
>
> I also currently struggle to find relevant news articles related to cryptocurrency. I would like to be able to access articles from multiple sources in one place, so I can view them at

> a glance and assess overall sentiment.

Me: What platform do you normally trade on?

> **Client**: I normally trade and perform all my analysis on my desktop computer. I wish for the application to be accessible through a website interface, so I can access it from wherever I am.

Me: What is important to you when visiting a website?

> **Client:** For me, speed and performance is very important. If a website takes too long to load, I will normally not bother waiting and just close it. The website you make should be as fast as possible to load and also responsive.

Me: Are there any other features you might like?

> **Client**: I also have many friends in the field. I would like the web application to support multiple users signup up and creating accounts. I would then like to be able to comment on articles that feature on the website, and also view other users comments. This will allow me and my friends to communicate and speculate together.
>
> I would also like to be able to view some basic information about the top coins from the website, such as the price and how they are performing.

CURRENT SYSTEM

After my interview with my client, I made a flowchart of how he currently uses Twitter and news publications to research cryptocurrencies to invest in.

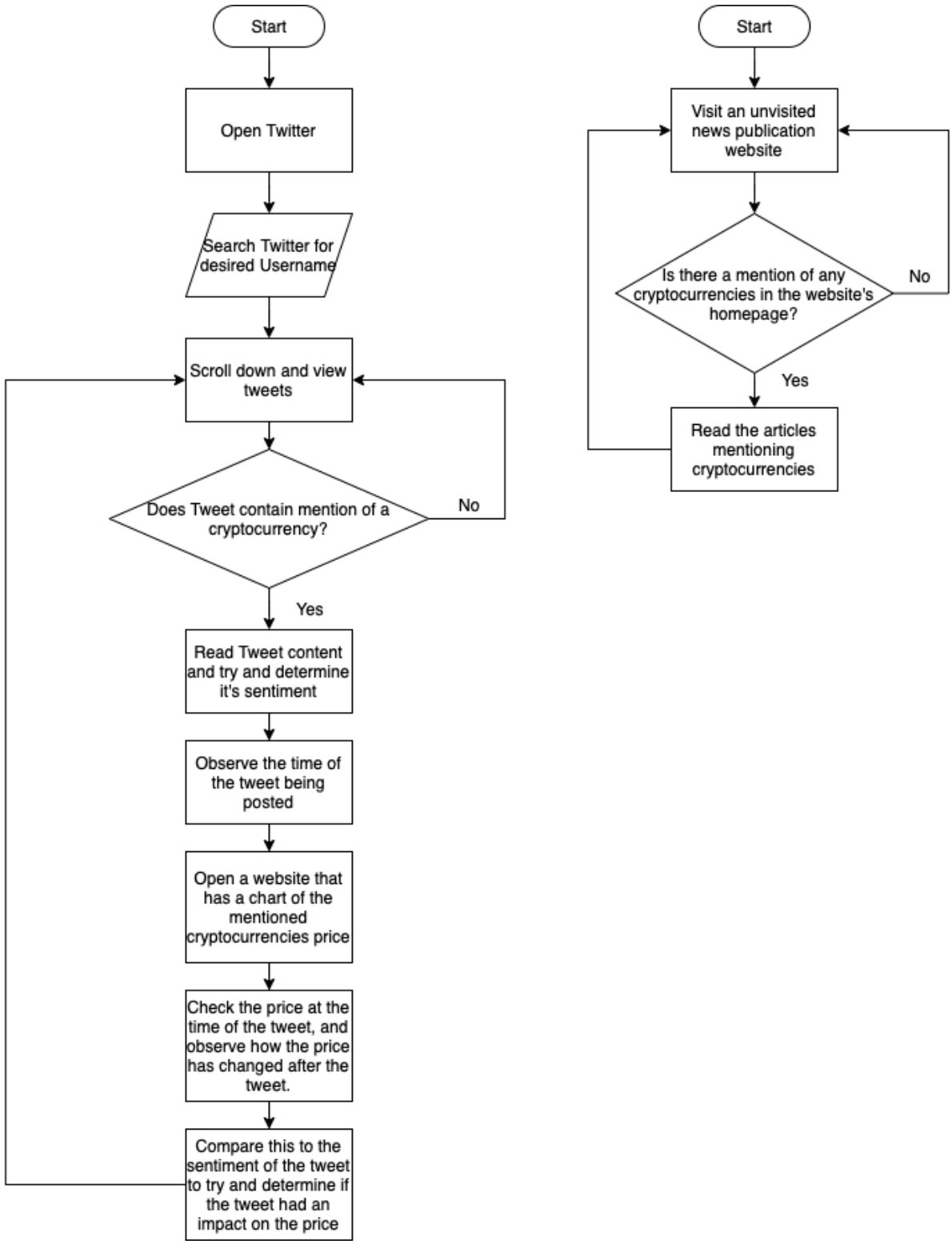

Figure 3: Flowchart of my clients process

As you can see, it is rather inefficient and involves him manually going through many tweets and articles, sometimes with no results. My proposed system should aim to resolve this inefficiency, and should save him a lot of time.

PROPOSED SYSTEM

Following the interview with my client, I created a proposal to send to him that aimed to satisfy his requests. You can see the proposal below:

> **Cryptica Social Media Analysis Application Proposal**

I am proposing a tool called Cryptica to help you meet your analysis needs. Cryptica will be a website application that consists of several main parts.

The main part of the application will be 2 separate analysis pages. These will both consist of an input field for you to input any user's twitter handle. Upon entering a user's handle, one of the pages will display a list of their tweets mentioning Cryptocurrency. Then, you will be able to click on any of the tweets which will bring up further analysis, as well as a graph showing the impact the tweet has on the cryptocurrency market. It will also display the predicted sentiment of the tweet - whether or not the content of the tweet is positive or negative. You will be able to use this to make an informed decision on whether a user has influence in the cryptocurrency space, and whether to follow what they are saying or not.

The second analysis pages will be for general analysis of a Twitter user. After entering a user's handle, you will be able to view a collection of graph and metrics about the user. This will include metrics such as what time they are typically active on Twitter, as well as what device they typically use Twitter on. This will allow you to gain an insight into their tweeting habits, further aiding your analysis.

Cryptica will have support for account authentication - you and your friends will be able to create accounts and stay logged in between sessions, meaning you can have direct control over who can use your tool. To use either of the analysis pages, you will be required to login. This login system means you can have fine access control over who can use your tool.

Next, the news section. Cryptica will store excerpts of news articles from many high profile news sources, and will allow you to easily view headlines relevant to the cryptocurrency market from many sources in one place. There will be a main news page that displays a title and preview of all the news articles in the database. You will be able to click on any of them

and it will take you to a page dedicated to the article. On this page, you will be able to view the full details about the article, and click through to visit the original article. You will also be able to post a comment using your account on the article, and view other user's comments.

Finally, I will include a page that displays a summary of the cryptocurrency markets and the top coins. You will be able to see a list of the top coins by market cap, and click on any of them to view a graph of their price. This will let you view at a quick glance how certain coins are performing, as well as some key metrics with the coin.

This will be implemented using a client application that interacts and works with an API server. This can be hosted on a number of free hosting services at no cost or difficulty to you.

Please get back to me and let me know what you think, and if you would like any changes. - Arthur

I also attached a copy of the following flowchart, highlighting how my application will work:

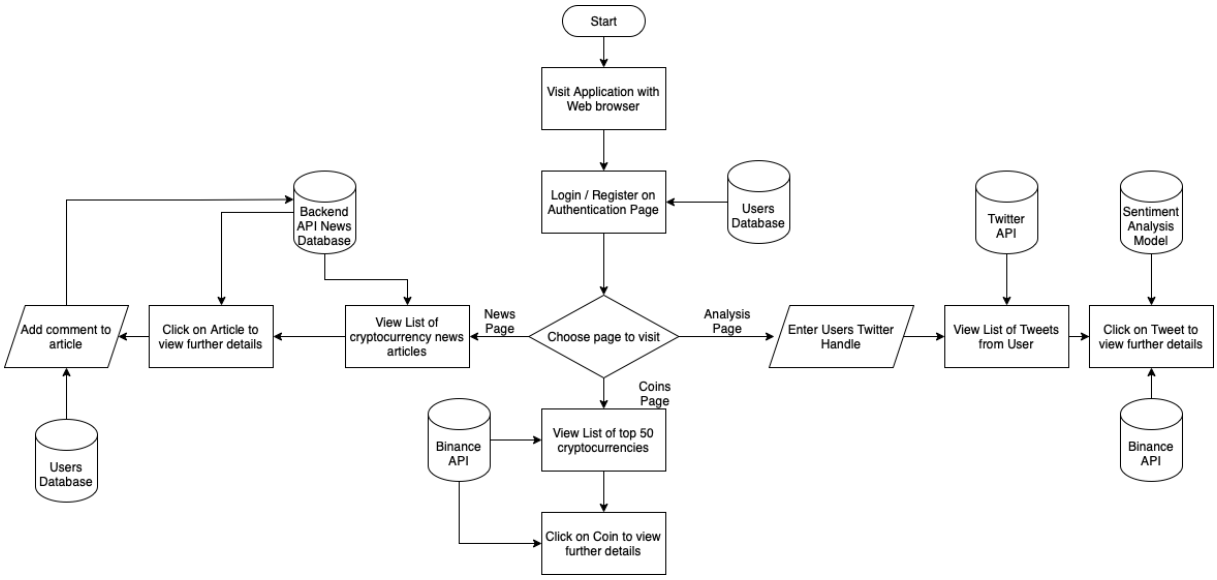

Figure 4: Flowchart showing how the user will interact with the program

My client responded approving the proposal, and requested no changes.

OBJECTIVES

1. There should be a publicly accessible web application that allows the client to access the etc

2. There should be two main components to the web application, a front end client and a back end API. The front end client should interact with the API and should be what the user interacts with. The API should handle all the fetching/processing of data, as well as any other functionality such as database management with CRUD.
3. The API should be capable of securely authentication users for the front end app. My client has specified that he would like anyone to be able to login and register for an account, so that he can share the application with his like-minded friends. Therefore, the API should be capable of handling multiple concurrent users having accounts, and should be able to authenticate and distinguish between them.
4. The API should interact with the front end client application to ensure that users remain authenticated between sessions. Users should be able to login and then have to not enter their password again for a reasonable amount of time. This should be done using JSON Web Tokens. These should be created and signed by the API server using cryptography, and stored in the browser storage.
5. The RSA algorithm should be used to sign the JSON Web Tokens. To do this, I will need to have an RSA key. Part of the program should be able to create RSA keys for use in this functionality.
6. The users data and passwords should securely be stored in a database. A secure, modern password hashing algorithm should be used, that uses hash salting to protect against attacks.
7. The API should be capable of fetching and processing a specified users tweets from Twitter's API. It should be able to perform sentiment analysis on the tweet's content, and return the information to the user.
8. The API should have a database table that stores a collection of recent relevant news articles. The frontend client should then be able to display these articles for easy access. The news articles should ideally come from a variety of sources through web scraping. Only a brief excerpt of the article needs to be stored and displayed - to read the full article the users should be directed to the original site. Alternatively, the news articles should be fetched from an existing third party API that offers a service.
9. There should be a page that displays a list of the top 50 coins by market cap. It should display live data showing the price and other statistics about the coins. You should be able to click on any of the coins and it should take you to another page, showing further information about the coins performance. This should include a graph of the coins performance over time, and a brief description of the coin. In addition, on the specific coin page it should show a list of relevant articles stored in the database relating to the

coin. If no such articles are found, it should not display any.

10. Logged in users should be able to comment on any of the news articles, and anyone should be able to view said comments. Admin accounts should be able to delete any users comments, and users should be able to delete their own comments. The API should be able to distinguish between users and administrators.
11. There should be a basic profile functionality. Users should be able to view a users profile, and view information such as all their historic comments on articles.
12. The application should be secure against malicious parties. It should not be vulnerable to common flaws such as SQL or XSS (cross site scripting) injection, and users should not be able to bypass authentication methods implemented, e.g. viewing pages that are behind an authentication wall.
13. The application should contain analysis page for users tweets, that allows someone to input a users Twitter username. Then, they should be able to view a list of tweets, and should be able to see information on how the tweet has impacted the cryptocurrency market. It should display a candlestick graph that displays the price of the relevant cryptocurrency before and after the tweet. This page should also show the predicted sentiment of the tweet - whether the tweets content is positive or negative.
14. The analysis page should also offer some basic analysis on the user's Twitter account as a whole. It should be able to produce a heat map of the time the user is typically active on Twitter, based on the time the user has tweeted previously. It should also display what device the user uses in the form of a pie chart, for example if the user is tweeting from an iPhone or from a computer.
15. The tweets analysis page should be able to perform some basic sentiment analysis on the user's tweets contents. It should attempt to estimate whether a tweet is positive or negative, and this should then be displayed to the user. For this, a neural network should be implemented using a Python deep learning library such as TensorFlow. The neural network should aim to have an accuracy of around 75%+. This objective is ambitious and primarily an extension that I should complete if I have enough time. Failing that, it should use an existing third party API for analysing sentiment, rather than creating my own sentiment model.
16. The website should be fast to respond. This can be measured using Google's Lighthouse page score metrics, which is a service that returns a value on how fast the page performs. I want to aim for a score of 90-100, which is considered 'excellent'.

OBJECTIVES COMPLEXITY AND LIMITATIONS

My objectives are of high complexity, and will require learning and working with many different elements. It will require me to use several different APIs, including Twitter's API and a Cryptocurrency price API.

Security

Security is a big part and focus of my project. I will attempt to ensure my program is secure against all likely attacks. This is very difficult however, as the field of cyber security is constantly changing and evolving. This means that there are constantly new attack methods being developed, making it near impossible to claim an application is "100% secure". Instead, I will just attempt to make my program as secure as feasible given my limited time and expertise.

API Data Resolution

As part of my project I will be processing data from several third party APIs, including Twitter's API and a Cryptocurrency API. These APIs normally impose restrictions on the quality of data freely available. I will be limited by what data I can freely access.

With my project I will be required to process very high quantities of data frequently. It will be important that my code is efficient and does not have any bottlenecks. This will add a high level of complexity.

Neural Networks

Neural Networks are a complicated topic heavy on maths. As part of my project I will attempt to understand and implement several complex machine learning algorithms. As mentioned before I aim to produce a sentiment analysis model capable of classifying sentiment with an accuracy of above 75%. However, neural networks are a new field to me so I must accept that this might not be possible with my limited time. If I fail to produce a working model, I will instead resort to using a third party API to perform sentiment analysis, which should achieve the same end result.

User Interface Design

Whilst the User Interface is important, creating one with CSS is difficult and time consuming. I shall instead be focusing the majority of my time with building functionality to my application. I shall also be using a CSS utility library called Tailwind, which shall help speed up development.

News Article Scraping

Despite a lot of websites looking visually similar, they are all composed of very different HTML. This poses a challenge when attempting to scrape websites for news articles, as it is hard to make a program that is capable of scraping articles from a large variety of sources. For this reason I will likely be using an external News API instead of scraping. I will likely be limited by what API is freely available.

AUTHENTICATION

Authentication will be a large part of my project. Authentication is the process of verifying an identity, and ensuring that a user interacting with my system is who they claim to be. Once a user has authenticated with my server initially, I need a way to keep them logged in and verify that the user is who they claim to be, without them having to enter their password each time. I have researched several of the most popular authentication methods and compared their pros and cons to help me decide on which authentication method to implement. You can see my research and comparisons below.

HTTP Basic Authentication

HTTP Basic Authentication is the simplest form of authentication that is built into the HTTP protocol. It involves sending a header containing login credentials with each request made to a website. The header will look like the following: `Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=`. `dXNlcm5hbWU6cGFzc3dvcmQ=` is `username:password` base64 encoded to form a string that can be sent with HTTP requests. The receiving server will then compare the username and password value sent in the request with a value in a database.

This authentication method is stateless, so the username and password must be supplied with each request to the server.

Pros

- Stateless
- Easy to implement
- Requires little computing power, fast
- Supported by most browsers

Cons

- Credentials are sent unencrypted to the server, therefore HTTPS essential
- Hard to log users out / invalidate credentials
- Credentials must be sent with every request
- Requires storing passwords in plaintext

HTTP Digest Authentication

HTTP Digest Authentication is a variant of HTTP Basic Authentication that addresses the lack of encryption when sending the password. Instead of sending the base64 encoded password in cleartext, it is hashed before being sent to the server. This means that if it is encrypted, it is much harder to extract the original password.

Pros

- Same as HTTP Basic Authentication
- Passwords sent encrypted

Cons

- Credentials must be sent with every request
- Hard to log users out / invalidate credentials
- Password hashing algorithm must be ran on the client and server, limiting options

Session Based Authentication

With session based authentication, the user's authentication state is stored on the server typically in some form of database. Rather than requiring the user to supply a username and password with each request, after logging in once the server creates a session object. This can be then stored in a database, and a session ID can be sent back to the client to store in the browser. This session ID is then sent with all future requests, and is then verified by the server upon receiving a request.

Pros

- Only requires sending credentials once
- Widely supported with most popular web frameworks
- Allows invalidation of sessions - can remove session from database and force user to log out

Cons

- Stateful - requires implementing a session database. The server needs to keep track of all sessions generated, which requires additional computing power
- If a user's session ID is intercepted and stolen, an attacker could perform malicious acts on behalf of the user

Token Based Authentication

Token based authentication has some similarities with session based authentication, but differs in the use of a stateful database. With token based authentication, upon valid credentials being supplied to the server, the server generates and signs a token. This token is then stored by the client and sent with subsequent requests. Then, the server can simply verify the tokens signature to determine if it is valid. This means the server does not need to keep track of tokens generated.

Pros

- Stateless - the server does not need to keep track of tokens generated
- Low overhead, with little computing power needed

- Rising popularity over recent years, with many companies adopting their use. Lots of documentation online
- Tokens are compact and typically small in size

Cons

- Difficult to invalidate tokens, tokens are only invalid when they expire
- Token stored in cookies/browser storage, which can sometimes be exploited by attackers

OAuth

OAuth/OAuth2/OpenID are a form of single sign-on (SSO) that allows users to authenticate using an existing account from applications such as Google, Facebook and Apple. They allow you to create accounts and login to new websites using your existing account on another service. This means there is no need to create or store new passwords, and the other service handles all credential storage. OAuth is very popular, and many millions of people use it on a daily basis. You will typically see an option when creating an account to “Login with Google” or another service.

Pros

- Improved Security
- No need to store usernames / passwords
- Easy experience for the user and fast
- Uses external applications existing authentication infrastructure

Cons

- The application is dependent on external services
- Requires the user to have an account on a configured service
- Difficult to implement, involves working with many different services

Conclusion

After my research, I have decided to implemented a form of Token Based Authentication with my application, specifically JSON Web Tokens. I have researched JSON Web Tokens further,

and you can read further about them in the design section of this document.

USER IDENTIFICATION

My client has specified that he would like other users to be able to access and use the application alongside him. He however is the main user and will have admin privileges over the system, allowing him to moderate and maintain the application.

The secondary users will be other likeminded retail investors who use social media analysis to inform their cryptocurrency trading decisions. The typical user is tech savvy, and familiar with web applications like this. That being said, the program will need a intuitive user interface to allow new users to navigate through the application and use the tools. Ideally the application should function as a hub for social media cryptocurrency analysis. There will be a social aspect as well, with users able to create accounts and comment on news articles.

This creates its own set of problems, with moderation required. My client as mentioned before will have admin privileges allowing him to manage comments and users. In addition there will be a basic comment filter in place, that attempts to filter out inappropriate comments from being published.

There will be a guide on the homepage that will inform users how to use the application. This will be an easy way to help users understand how the program works, and what it does.

TECHNICAL SOLUTIONS

Next.js

Next.js is what I am going to be using to build my frontend application. Next.js is a JavaScript framework built upon React for developing web applications. React is one of the most widely adopted frameworks. According to Statista, React is used by over 40% of website developers worldwide. Next.js is also incredibly popular, with it being downloaded over 2.3 million times each week (source NPM). Next.js is a full stack framework, and is capable of handling both the front and backend of an application. I will primarily be using it for the front end, but it is useful having the option of using it for any backend functionality if needed. Next.js is incredibly flexible, and has an excellent developer experience. The majority of a Next.js application consists of components. Components are basic functions that return JSX. JSX is a special

syntax that looks like HTML, but can also contain JavaScript code. Components are then rendered on the page by either the client or the server, and turned into HTML. Next.js is incredibly fast, and can render components before hand on the server. This helps result in a seamless user experience with no loading times. In addition, rendering components on the server beforehand is excellent for Search Engine Optimisation (SEO), which is important when considering page rankings on search engines such as Google. Next.js also has a huge developer community, which provides excellent documentation, guides, and third party module extensions.

FastAPI

For my backend API, I chose to use a Python powered server. This is because I am experienced with using Python for data handling and processing, which will be a large part of my API server. Python is also very suited towards machine learning and artificial intellegence which will feature in my project. Python has a variety of web server modules available, however for my project I am choosing to use a Python module called FastAPI. My primary reason for choosing FastAPI is it's performance. It is a lot faster than other Python alternatives, such as Flask and Django, as seen in the table below.

Responses per second at 20 queries per request, undefined (24 tests)

Rnk	Framework	Performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	goji	14,797	100.0% (25.1%)	0	Mcr	go	non	non	lin	My	lin	raw	rea
2	falcore	14,569	98.5% (24.7%)	0	Mcr	go	non	non	lin	My	lin	raw	rea
3	gin	14,444	97.6% (24.5%)	0	Mcr	go	non	non	lin	My	lin	raw	rea
4	fastapi	12,774	86.3% (21.7%)	0	Mcr	Py	non	non	lin	Pg	lin	raw	rea
5	starlette	12,740	86.1% (21.6%)	0	Plt	Py	non	non	lin	Pg	lin	raw	rea
6	sanic	12,688	85.7% (21.5%)	0	Mcr	Py	non	non	lin	non	lin	raw	rea
7	express	11,067	74.8% (18.8%)	0	Mcr	JS	non	non	lin	Pg	lin	ful	rea
8	martini	10,970	74.1% (18.6%)	0	Mcr	go	non	non	lin	Pg	lin	raw	rea
9	express	10,529	71.2% (17.9%)	0	Mcr	JS	njs	non	lin	Pg	lin	ful	rea
10	flask-raw	6,542	44.2% (11.1%)	0	Mcr	Py	mei	non	lin	My	lin	raw	rea
11	flask-raw	4,961	33.5% (8.4%)	0	Mcr	Py	tor	non	lin	My	lin	raw	rea
12	express	4,463	30.2% (7.6%)	0	Mcr	JS	non	non	lin	My	lin	ful	rea
13	express	3,941	26.6% (6.7%)	0	Mcr	JS	njs	non	lin	My	lin	ful	rea
14	hapi	2,536	17.1% (4.3%)	0	Mcr	JS	non	non	lin	Pg	lin	ful	rea
15	pyramid	2,448	16.5% (4.2%)	0	ful	Py	non	mei	lin	Pg	lin	ful	rea
16	pyramid	2,426	16.4% (4.1%)	0	ful	Py	non	mei	lin	Pg	lin	ful	rea
17	hapi	2,356	15.9% (4.0%)	0	Mcr	JS	non	non	lin	My	lin	ful	rea
18	hapi-nginx	2,190	14.8% (3.7%)	0	Mcr	JS	non	non	lin	Pg	lin	ful	rea
19	flask	1,773	12.0% (3.0%)	0	Mcr	Py	mei	non	lin	My	lin	ful	rea
20	django	1,643	11.1% (2.8%)	0	ful	Py	non	mei	lin	Pg	lin	ful	rea
21	bottle	1,614	10.9% (2.7%)	0	Mcr	Py	tor	non	lin	My	lin	ful	rea
22	django	1,543	10.4% (2.6%)	0	ful	Py	non	mei	lin	My	lin	ful	rea
23	bottle	–	Did not complete	–	Mcr	Py	mei	non	lin	My	lin	ful	rea
24	bottle-raw	–	Did not complete	–	Mcr	Py	mei	non	lin	My	lin	raw	rea

Figure 5: Source: https://christophergs.com/python/2021/06/16/python-flask-fastapi/

FastAPI is also lightweight, and unlike some of the other libraries does not come with lots of

features I do not need. This will all help to keep my application fast, and performance high. FastAPI also has very similar syntax to Flask, another Python module that I have experience with. This should help ensure an easy development experience.

TensorFlow and Neural Networks

TensorFlow is a Python Library that can be used to create Deep Learning models, created by Google. TensorFlow has extensive documentation, and many abstraction layers, meaning it is friendly for beginners to develop with. I plan on using TensorFlow to create a sentiment analysis classifier, to use on tweets to calculate how positive or negative the content is. TensorFlow offers many several corpuses which will be suited for my project, including a library of 50,000 IMDB reviews categorised by rating. This will allow me to use the reviews to train a model to identify positive and negative text, which will be applied to my API.

Neural Networks are a type of machine learning algorithm loosely modelled on the human brain. They allow us to identify and classify data and patterns based on raw input. They consist of nodes, with typically an input layer, then one or more hidden layers, followed by an output layer. Each node connects to other nodes, and has a specified weight, bias and threshold. If the output of a node is above the threshold, the node is activated and data is sent to the next layer. Neural networks improve their accuracy over time by using training data to learn and fine tune their parameters. Once trained to a suitable level, they are able to consistently classify data at a quick rate.

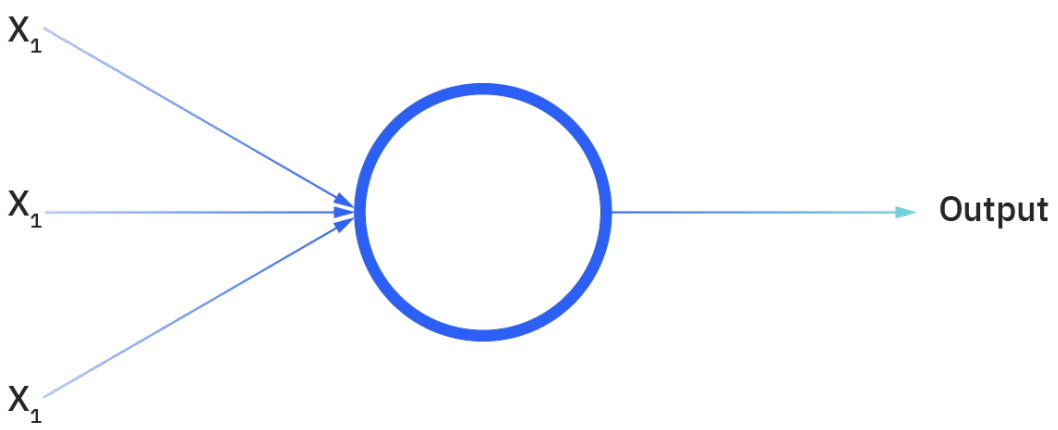

Figure 6: Diagram of a very basic neural network, from IBM's website.

I have also decided that my project will be using a Recurrent Neural Network model. Recurrent neural networks are a type of neural network that uses sequential data. They utilise training data to learn, but unlike a typical neural network they have “memory”, as they can take information from prior inputs to change the input and output. This means that the output is influenced by the order of the inputs

Twint

Twint is a Python library for fetching data from Twitter without using it’s API. Twitter has it’s own API that you can register for an account for, however it imposes strict limitations that made it unsuited towards my project. Twitter’s API does not allow you to fetch over a certain amount of tweets, and has harsh rate limitations preventing you from accessing too many tweets in a short period of time. My project relies on fetching a large quantity of tweets frequently, so I decided to use Twint instead. Twint is a library that fetches tweets by scraping from twitters website directly, as if it was a user viewing Twitter with a browser. This allows for a much quicker access to more data, and for a much better experience. Twint is open source, however is rather poorly maintained. In my testing, I experienced a variety of bugs and flaws with the package, stopping me from accessing the data that I required. However, I have created a fork of the package which I have modified to fix the bugs I encountered, fixing the package for myself.

Binance API

I am choosing to use Binance’s API to get Cryptocurrency data. According to coinranking.com, Binance is the most popular Cryptocurrency exchange, with 24 hour trade volumes typically reaching almost 10 billion dollars. Binance also lists and offers data on a very large number of markets, at time of writing being 1229. This will allow me to access a very vast quantity of data. In addition, Binance’s API for accessing data is free and offers generous rate limiting.

Argon2

Argon2 is a modern GPU resistant password hashing algorithm. It offers much better cracking resistance than other popular password hashing algorithms such as BCrypt, PBKDF2, and SCrypt. Argon2 is considered one of the best available in the industry, and is recommended

over other algorithms. It was selected as the winner of the Password Hashing Competition in July 2015, and is released under a Creative Commons License. It also has paramaters that allow you to configure the execution time, memory required, and degree of parallelism of the algorithm.

PostgreSQL

When choosing a database, I chose to use PostgreSQL. PostgreSQL is a highly stable database management system that is over 20 years old. PostgreSQL has many performance optimisations ensuring it is fast, and is a popular choice in enterprise applications. It is open source, and has integrations for most popular programming languages, including Python which is what I am using for my API.

TailwindCSS

TailwindCSS is a CSS Utility Library, which allows the use of utility CSS classes to rapidly build interfaces. I will be using it to save time when working on the user interface. Instead of needing to create my own CSS classes, it will allow me to use their pre defined utility classes within HTML class tags. TailwindCSS also supports custom theming; this will allow me to define a colour theme which can be changed at any point. This will let me update the colour theme across the entire site by just changing one variable, instead of updating each HTML class name manually.

DESIGN

OVERALL DESIGN

The program will consist of two main parts, a client website and an API server. The client server will interact with the API server to process and handle data and requests. The API server will connect many different APIs and Database Tables to the client. The table below highlights how the different parts could interact with each other.

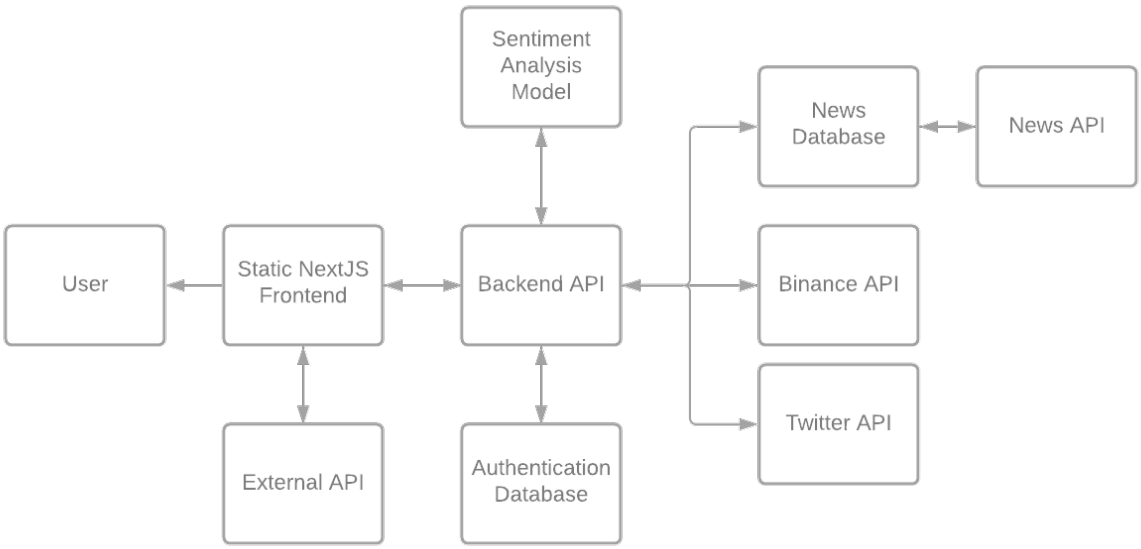

Figure 7: Diagram showing how the different parts could interact with eachother

FRONTEND PAGES

Below is a table containing a list of planned pages in the client frontend, and a description of what the function of each one is:

Page	URL Slug	Purpose
Homepage	/	This will be the page the user is greeted with upon first visiting the site. It should contain a brief description of how the site works and what it does, and should contain a set of frequently asked questions.
News	/news	This page should contain a list of news articles stored in the database. Users should be able to click on any of the articles which should take them to the articles page.

Page	URL Slug	Purpose
News Article	/news/[id]	This page will display the full details stored about an article. It will also have a comment field, that allows logged in users to comment and view other comments on the article. This page should also contain a link to the original article, so the user can refer to the site the article origins from if desired. Clicking on a user in the comment section should redirect the user to their profile page.
Coins	/coin	This page should display a table containing the top 50 cryptocurrencies ordered by market cap. The table should contain some details about the coin, such as the price and the 24hour change. It should also display a picture of the coin symbol. Clicking on any of the coins in the table should take you to a page dedicated to the coin.
Coin	/coin/[name]	This page should show some details about the specific coin. This should include a graph of the coin's price in the past year, as well as other statistics. Ideally this page should also have a description of the coin, if available. The page should also have a sidebar that contains links to any articles relevant to the coin stored in the database.
Login	/login	This page should contain a form enabling the user to login to the website. Clicking login should send a request to the API server which should verify the users password, and then redirect them to their account page. This page should not be accessible to logged in users.
Register	/register	The register page should offer users the ability to create an account. It should contain a form that asks for an email, name, and password. There should be a password requirement that ensures the inputted password is secure. Pressing register should send a request to the API that should attempt to register and sign in the user. This page should not be accessible to logged in users.

Page	URL Slug	Purpose
Analysis	/analysis	This page should contain an input field for the user to enter a twitter handle. Then pressing submit should send a request to the API server, which will return a list of tweets that should be displayed on the page. Then, the user should be able to click on any of the tweets which should bring up a graph and some analytics on the tweet, showing the impact it had on the cryptocurrency market. This page should only be accessible to logged in users.
Profile	/users/[id]	This page should act like a publicly accessible profile page. It should contain a list of comments the user has published on any of the news articles, and a link to each of them. User should be logged in to view this page.
Account	/account	The account page should only be accessible to logged in users. It should display some basic information about the logged in user, such as their name and email. It should also contain a button that enables them to logout.

API ROUTES

Below is a table containing a list of planned API routes, and a description of what the function of each one is:

Route	HTTP Methods	Purpose	Authentication
/api/hello	GET	Testing API route useful to verify that the server is running. Should always return “Hello World”.	None
/api/users/	GET	Should return a list of users from the database	Admin Only

Route	HTTP Methods	Purpose	Authentication
/api/users/	POST	Creates a user from the data supplied with the POST request	Admin Only
/api/users/{ID}	GET	Should return the specified user's details	Admin Only
/api/users/{ID}	PUT	Allows modifying the specified user by providing new details in the PUT request	Admin Only
/api/users/{ID}	DELETE	Should delete the user specified from the database	Admin Only
/api/users/count	GET	Should return a count of the number of users	Admin Only
/api/users/admin	GET	Should return a list of administrator users	Admin Only
/api/users/{ID}/profile	GET	Should return a list of comments from the specified user	Logged in Users Only
/api/auth/login	POST	This is responsible for logging in a user. Should return a JSON Web Token if the supplied authentication details are correct. If the supplied details are incorrect should return an error	None
/api/auth/register	POST	This should be for creating accounts. This should take the supplied POST data and attempt to create an account if one with matching details does not already exist. It should also return a JSON Web Token if the account has been created successfully, if not an error.	None

Route	HTTP Methods	Purpose	Authentication
/api/auth/me	GET	This should return a status 200 if the user is successfully logged in. If not, it should return an error. This is used to check if a user is logged in.	Logged in Users Only
/api/auth/edit	PUT	This should allow a user to update their own account by supplying the PUT request with new data.	Own User Only
/api/news	GET	This should return a list of the news articles in the database ordered by date. It should not return the full articles, just what is needed for the news index page.	None
/api/news	POST	This should allow the creation of new articles through POST request data.	Admin Only
/api/news/comments	GET	This should return a list of all the comments in the database.	Admin Only
/api/news/{ID}	GET	This should return the full details about a specified news article, used by the client to display.	None
/api/news/{ID}/comments	GET	This should display a list of comments on the specified article, ordered by date.	None
/api/news/{ID}/comments	POST	This should allow the creation of new comments, by supplying the comment content in the POST data.	Logged in Users Only
/api/news/{ID}/comments/{COMMENT_ID}	PUT	This should allow the user to edit their comment by supplying new data in the PUT request. Users should only be able to modify their own comments.	Author/Admin Only

Route	HTTP Methods	Purpose	Authentication
/api/news/{ID}comments/{COMMENT_ID}	DELETE	This should delete the specified comment on the specified article. Only the author of the comment or an admin should be able to delete a comment.	Author/Admin Only
/api/news/search	POST	This should search the database for news using paramaters supplied in the POST request, and return any results as a list.	None
/api/twitter/search	POST	This should cause the API to return a list of tweets for a user specified in the POST request, from Twitter's API.	Logged in Users Only
/api/crypto/{TICKER}/{TIME}	GET	This should return a set of price data for a specified cryptocurrency at a specified time. It should return the data in a list in the OHLC format (Open, High, Low, Close).	Logged in Users Only

INPUT PROCESS STORAGE OUTPUT CHART

Input	Process	Storage	Output
Login: Username, Password	Authenticate Login Credentials and generate signed JWT	Credentials supplied hashed with same settings and hash in database, stored in temporary variable Compared against hash in database RSA Private Key stored in environmental variable for signing JWT	Depending on success, either: Signed JWT containing authentication data Error Message
Register Account: First name, Surname, Email, Password	Create new User Users supplied password hashed	User added to Users table, with supplied inputs and hashed password	User Successfully created User creation Unsuccessful
Create Comment: News ID, Content	Creates comment Check content for potentially offensive content	Comment added to Comments Table	Comment added Successfully Comment added Unsuccessfully
Delete Comment: Comment ID	Check comment creator matches who initialised the request, OR if the user has admin privileges	Comment deleted from Comments Table	Comment removed Successfully Comment removed Unsuccessfully
Twitter User Search: Username	Download username's Tweets from the Twitter API	Store tweets in temporary array	Return array of tweets Return error of none

FORM STRUCTURE

As my project is a website based application, the user will interact and supply data to my program through HTML forms. There can then be a JavaScript function that takes the data from the forms and makes a request to my API server containing the form data.

Login Form

The login form will consist of an email and password field, with a submit button. The password field will have the `type="password"` attribute, which means that the input will remain hidden when entered, and will display as a • instead. E.g. when 12345678 is entered, it will be displayed as ••••••••. This is a security feature implemented in HTML, that means that even if someone can view your screen when you are entering your password, your password remains unknown. Upon the submit button being clicked, a JavaScript function will be called. This function will take the inputs inside the form, and submit a POST request to my API server login endpoint containing them. From there, my server will process and verify the data, and will return a status code. My website frontend will then display either a success or error message depending on the status code returned.

Registration Form

This form will consist of a First Name field, a Surname field, an Email field, a password field, and a submit button. Again, the password field will have the `type="password"` attribute. Upon clicking submit, another JavaScript function will take the form inputs and submit a POST request to my API Server Register endpoint.

DATA DICTIONARY

My main application will consist of 3 tables, Users, News, and Comments. The users table will store the details required to authenticate users and provide basic profile functionality. The news table will store a list of news articles scraped to be displayed on the news article page, and on the relevant cryptocurrency page. The comments table will store comments by the users on certain news articles, and will reference both the users and news table. This ensures that data is not unnecessarily repeated. Each user only features once, and can correspond to multiple comments. Each news article can have multiple comments associated with it.

Below you can see a table that has a plan of the different columns that my three tables will have. The column names are not final, and just serve as a description for now.

Column	Table	Data Type	Description	Example Data	Validation
ID	Users	serial	Auto generated primary key for user ID.	1	Required, Incremental
First Name	Users	varchar(255)	The first name that the user provides if they wish.	Arthur	Must be below 255 characters, A-Z only
Last Name	Users	varchar(255)	The last name that the user provides if they wish.	Robertson	Must be below 255 characters, A-Z only
Email	Users	varchar(255)	The email address of the user.	arthur@mail.com	Required, must be below 255 characters
Hashed Pass-word	Users	varchar(255)	The users password hash, salted and generated by the server.	$argon2i$v=19$m=16,t=2,p=1$dWRSS2o5dU84S3BrQVdBUA\$ir50+n9sxd1+qBs3kNaY/A	Required, must be below 255 characters and in argon2 hash format
Admin	Users	boolean	A boolean that states whether a user is an Admin which grants certain privileges.	true	Default is false. Only required if true.

Column	Table	Data Type	Description	Example Data	Validation
ID	News	serial	Auto generated primary key for the news ID.	378	Required, Incremental
Publication	News	varchar(255)	The publication/-source of the news article.	The Verge	Required, must be below 255 characters
Author	News	varchar(255)	The author of the article, if supplied.	Mitchell Clark	Must be below 255 characters
Title	News	varchar(511)	The title of the news article.	US banking regulators are looking to clarify crypto rules in 2022	Required, must be below 511 characters
Description	News	varchar(1023)	A summary of the news article, if supplied.	Three US agencies have issued a joint statement saying…	Must be below 1023 characters
URL	News	varchar(1023)	A link to the original news article.	https://www.theverge.com...	Required, must be below 1203 characters and a valid URL.
Image URL	News	varchar(1023)	A link to the feature image of the news article, if supplied.	https://cdn.vox-cdn.com/...	Must be below 1203 characters and a valid URL.

Column	Table	Data Type	Description	Example Data	Validation
Date	News	varchar(255)	The date of the article's publication in UNIX timestamp format.	1640950072	Required, must be below 255 characters and in UNIX timestamp format.
Content	News	varchar(1023)	Up to the first 1200 characters of the article.	One of them is already working to make banks…	Required, must be below 1203 characters.
ID	Comments	serial	Auto generated primary key for comment ID.	51	Required, Incremental
User ID	Comments	int	Foreign key, references a user in the USERS table.	1	Required, must be a valid USER_ID
News ID	Comments	int	Foreign key, references a news article in the NEWS table.	378	Required, must be a valid NEWS_ID
Date	Comments	varchar(255)	The date of the comments creation in UNIX timestamp format.	1640950072	Required, must be below 255 characters and in UNIX timestamp format.

Column	Table	Data Type	Description	Example Data	Validation
Content	Comments	varchar(2000)	The content of the comment that the user has inputted.	Oh no!	Required, must be below 2000 characters. Disallowed characters should be stripped.

ENTITY RELATIONSHIP DIAGRAM

A comment belongs to one news article and one user. An article and a user can both have many comments.

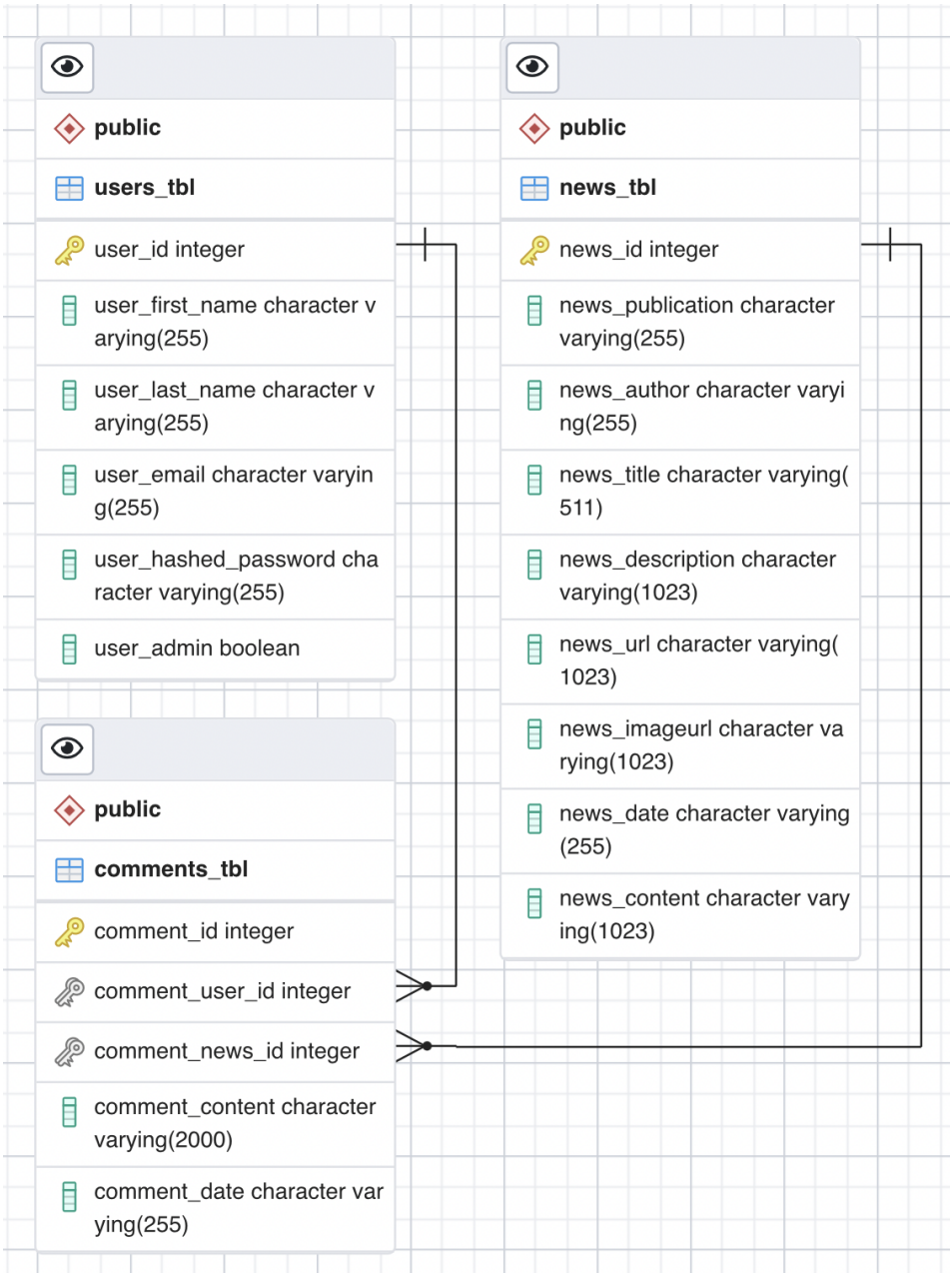

Figure 8: Entity Relationship Diagram for my tables

SQL QUERIES PLAN

This table shows a few examples of SQL queries I will be using. The dollar symbol followed by a number represents a variable. My project is using PostgreSQL, which is very similar to

MYSQL but offers some more features and better performance as mentioned previously. $n represents variable n.

Description	SQL Query
Select all entries from the specified table	SELECT * FROM $1;
Count entries in the specified table	SELECT COUNT(*) FROM $1;
Create a User in the Users table	INSERT INTO users_tbl (user_first_name, user_last_name, user_email, user_hashed_password, user_admin) VALUES ($1, $2, $3, $4, $5);
Delete item from specified table when variable matches	DELETE FROM $1 WHERE $2 = $3;
Update User by ID	UPDATE users_tbl SET user_first_name = $1, user_last_name = $2,user_ email = $3, user_hashed_password = $4, user_admin = $5 WHERE user_id = $6
Create Users Table	CREATE TABLE IF NOT EXISTS users_tbl (user_id serial PRIMARY KEY, user_first_name varchar(255), user_last_name varchar(255), user_email varchar(255), user_hashed_password varchar(255), user_admin boolean);
Create News Table	CREATE TABLE IF NOT EXISTS news_tbl (news_id serial PRIMARY KEY, news_publication varchar(255), news_author varchar(255), news_title varchar(511), description varchar(1023), news_url varchar(1023), news_imageUrl varchar(1023), news_date varchar(255), news_content varchar(1023))
Create Comments Table	CREATE TABLE IF NOT EXISTS comments_tbl (comment_id serial PRIMARY KEY, comment_user_id int, comment_news_id int, comment_content varchar(2000), comment_date varchar(255))
Drop Table	DROP TABLE $1;

Description	SQL Query
Get all News ordered by Date	SELECT news_id, news_title, news_publication, news_imageurl, news_description, news_date FROM news_tbl ORDER BY news_date DESC
Get news with keyword and limit	SELECT news_title, news_imageurl, news_publication, news_id, news_date FROM news_tbl WHERE UPPER(news_title) LIKE $1 OR UPPER(news_description) LIKE $1 OR UPPER(news_content) LIKE $1 ORDER BY news_date DESC LIMIT $2
Search item by column	SELECT * FROM $1 WHERE $2 = $3;
Get Comments and User info from News ID ordered by Date	SELECT comment_id, comment_user_id, comment_content, comment_date, user_first_name, user_last_name FROM comments INNER JOIN users ON comments.comment_user_id = users.user_id WHERE news_id = $1 ORDER BY date DESC
Insert an item into the news table	Insert into News: INSERT INTO news_tbl (news_publication, news_author, news_title, news_description, news_content, url, news_imageurl, news_date) VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
Create a comment in the comments table	INSERT INTO comments_tbl (comment_user_id, comment_news_id, comment_content, date) VALUES ($1, $2, $3, $4)
Get News and associated Comments by News ID	SELECT * FROM news_tbl INNER JOIN comments_tbl ON news_id = comments_news_id WHERE news_id = $1 ORDER BY comment_date DESC

CLASS DIAGRAMS

Below is a draft of some of the planned classes that I will be using in my program. I will be making use of Object Orientated techniques such as encapsulation and abstraction to

efficiently represent complex structures.

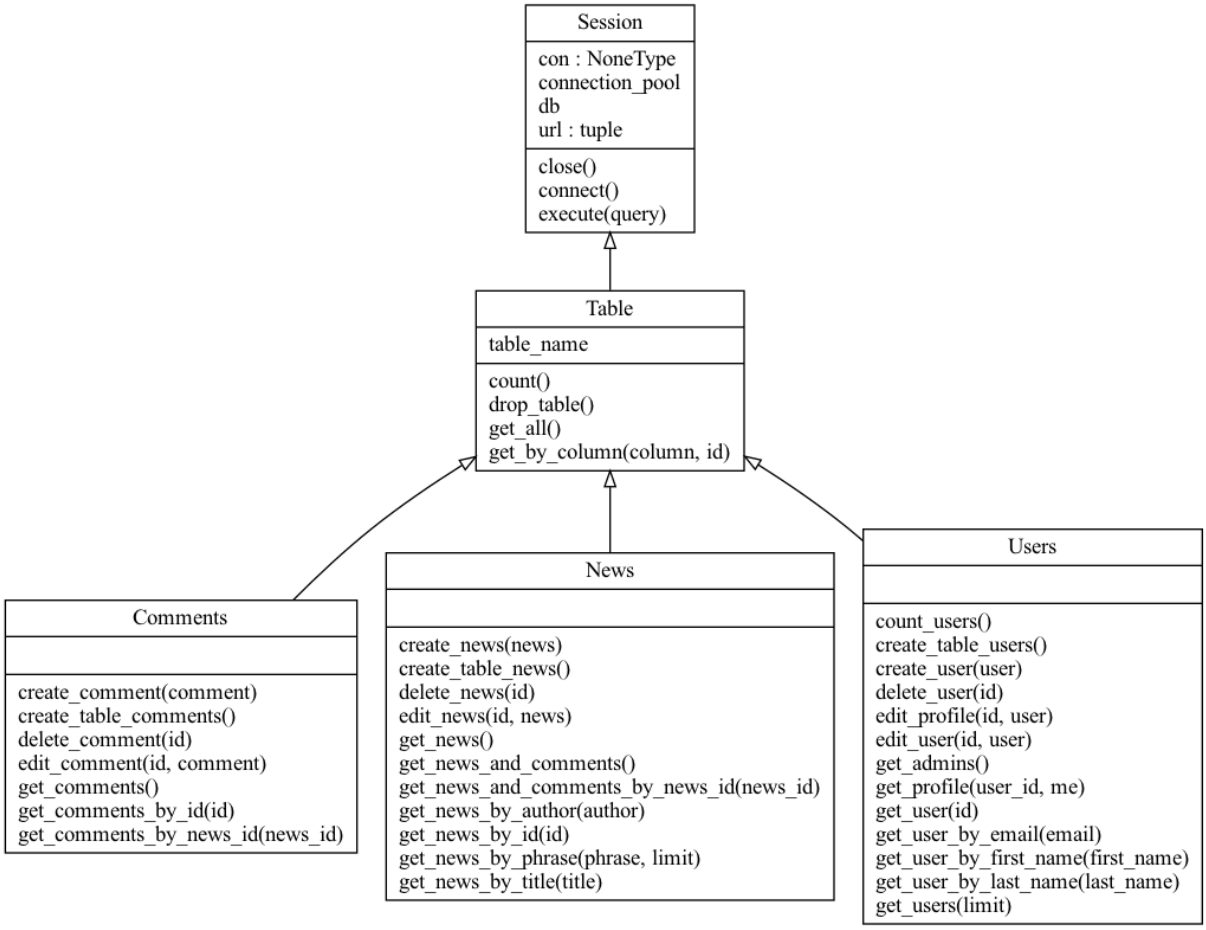

Figure 9: Class Diagram for the Database related classes

This is a UML diagram of the Database related classes. The Session class is used to manage the connection to the database, and access it at a low level. The Table class inherits the Session class, and counts some generic functions, such as a function to get a count of the number of rows in that table. The Comments, News, and Users class then inherits from the Table class, and passes the table name to the Table class. These classes contain more special purpose functions for interacting with the specific tables, however they can still access the general functions when needed from the Table class.

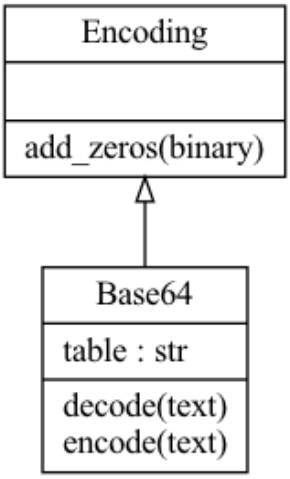

Figure 10: Class Diagram for the Encoding related classes

The above is the class for encoding Base64 into and from Decimal. It inherits from the class Encoding. This class exists in case I have a need to add any further encoding methods. Generic functions that typically feature in encoding such as recurisvely adding zeros (`add_zeros`) can feature here.

This UML diagram below contains the classes that will feature in the security section of my application. AccessToken is the class that will be used to create JSON Web Tokens, and Argon2 is the class that will be used to hash and verify passwords.

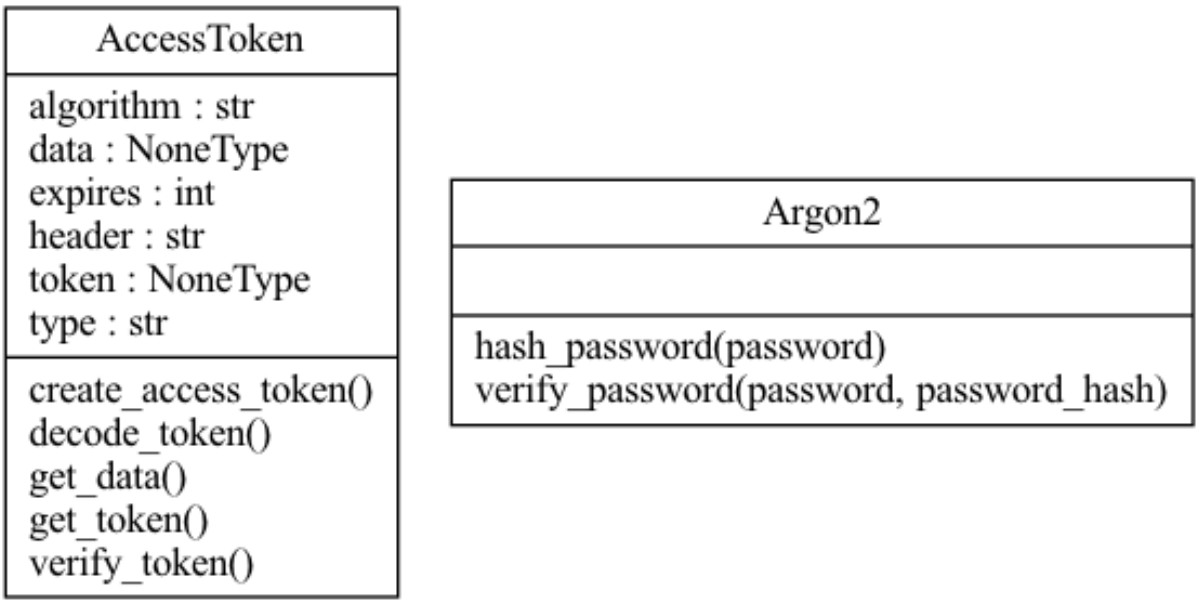

Figure 11: Class Diagram for the Security related classes

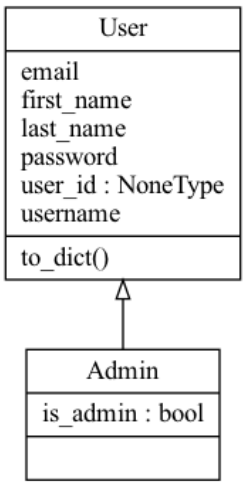

Figure 12: Class Diagram for the User model

This is a draft of how the User's can be represented with classes. The User class will have most of the methods and apply to most people using the site, and the Admin class will have all the same functionality with some added uses for Admins only.

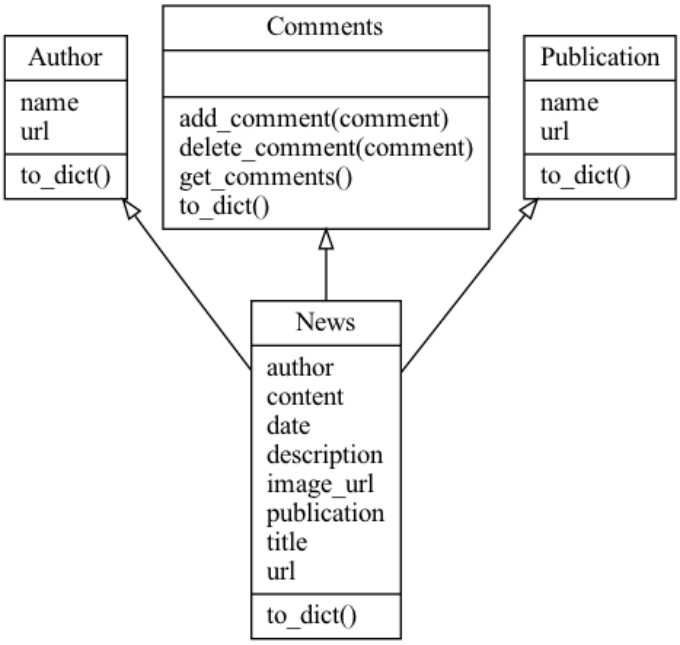

Figure 13: Class Diagram for the News Article related classes

Here the News class inherits from the Author, Comments, and Publication class. I propose that the comments Class has a data structure that allows comments to be removed and added.

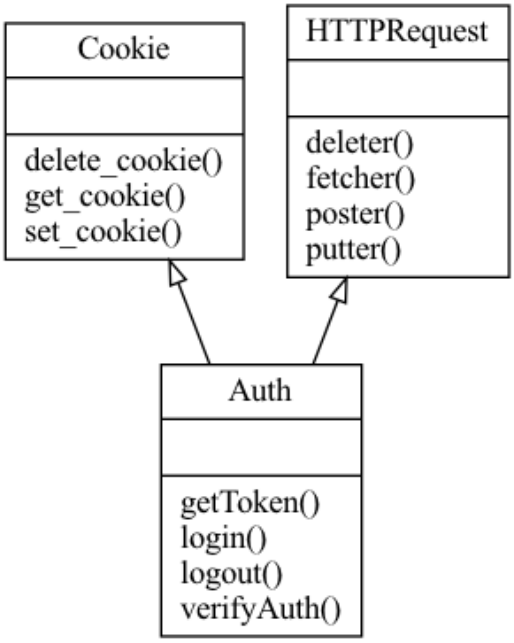

Figure 14: Class Diagram for the Front End Authentication

This diagram shows a class that will be used in the frontend to handle authentication. The Auth class will make use of methods in Cookie and HTTPRequest to send requests to the API server to generate JWT tokens, and then save the returned token as a cookie. It will also handles authenticated requests made to the server. Cookies need to be sent with requests that are made to the server so the server can verify the user's identity, so the Auth class will make use of both it's inherited classes to do this.

USER INTERFACE

When creating a mockup of how the user interface should look, two things were particularly important. Functionality, and accessibility. It was crucial that the user interface should be easy to use and efficient - not bogged down with unnecassary bloat like many large websites these days. Below is a wireframe of the header that will feature on all pages. The high contrast colours make it accessible, but still functional and aesthetically pleasing. The bar showing the cryptocurrencies and their respective prices should ideally scroll, enabling approximately 10 coins to be shown on loop. There should also ideally be an "account" / "sign in" button, perhaps where the "Search" button currently is. This can change depending on the user's login state.

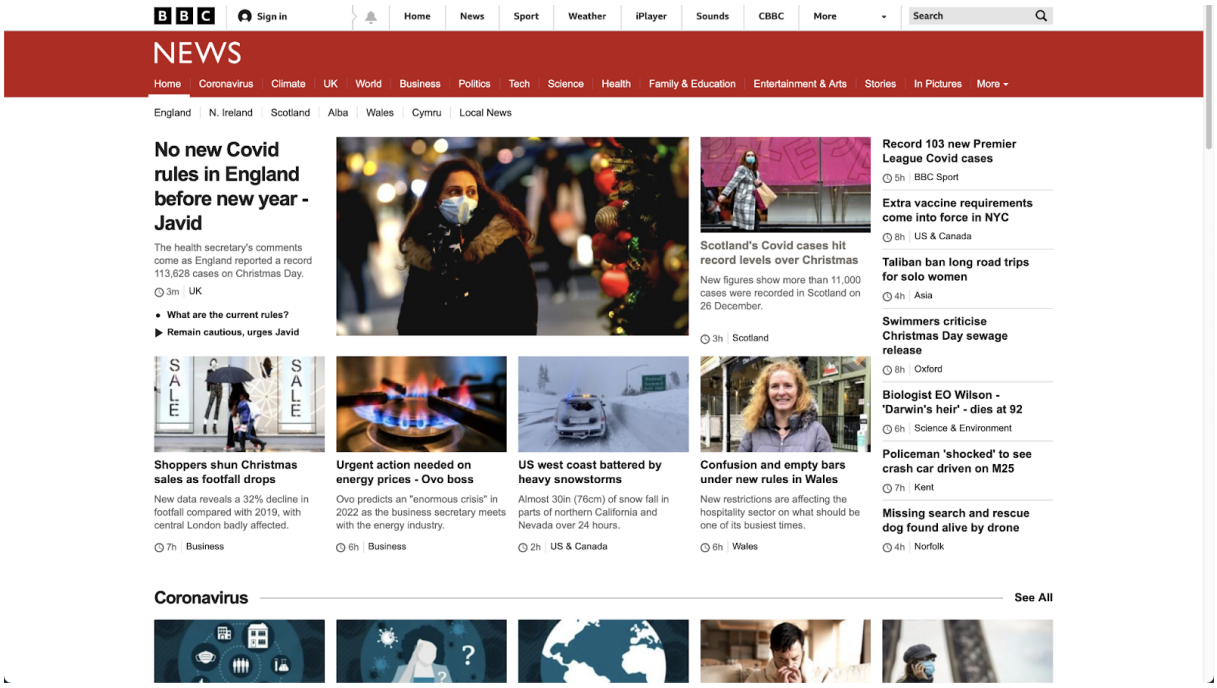

Figure 15: The inspiration for the design, BBC News

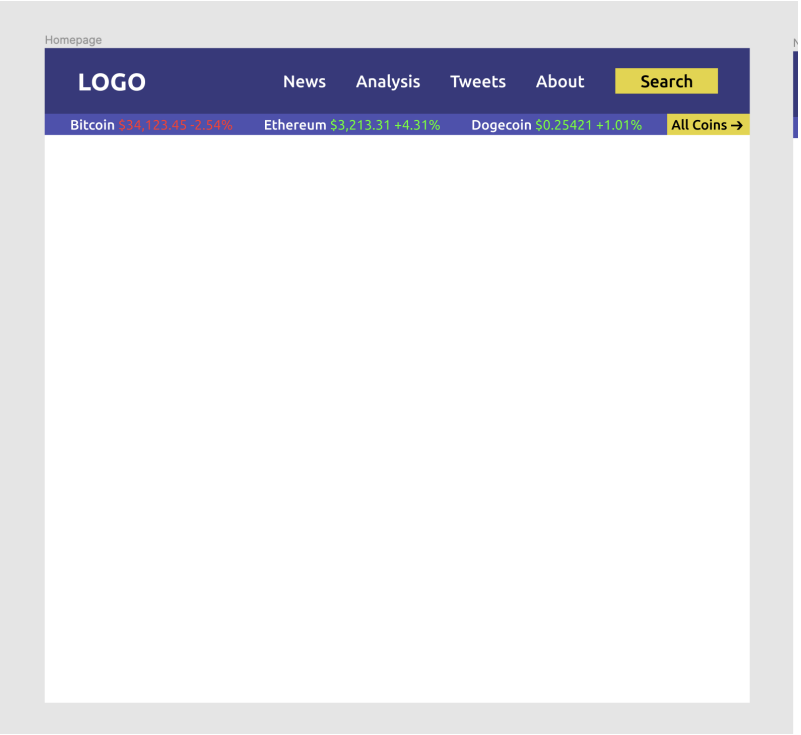

Figure 16: A mockup of how the header and a general page could look.

For the analysis page, there are 3 main parts. The input field for the user to search for someone's twitter handle, the tweet box that displays a list of the inputted user's tweets, and finally the analysis section that displays the price change, as well as the sentiment of the selected tweet. I put together a basic mockup of what this could look like.

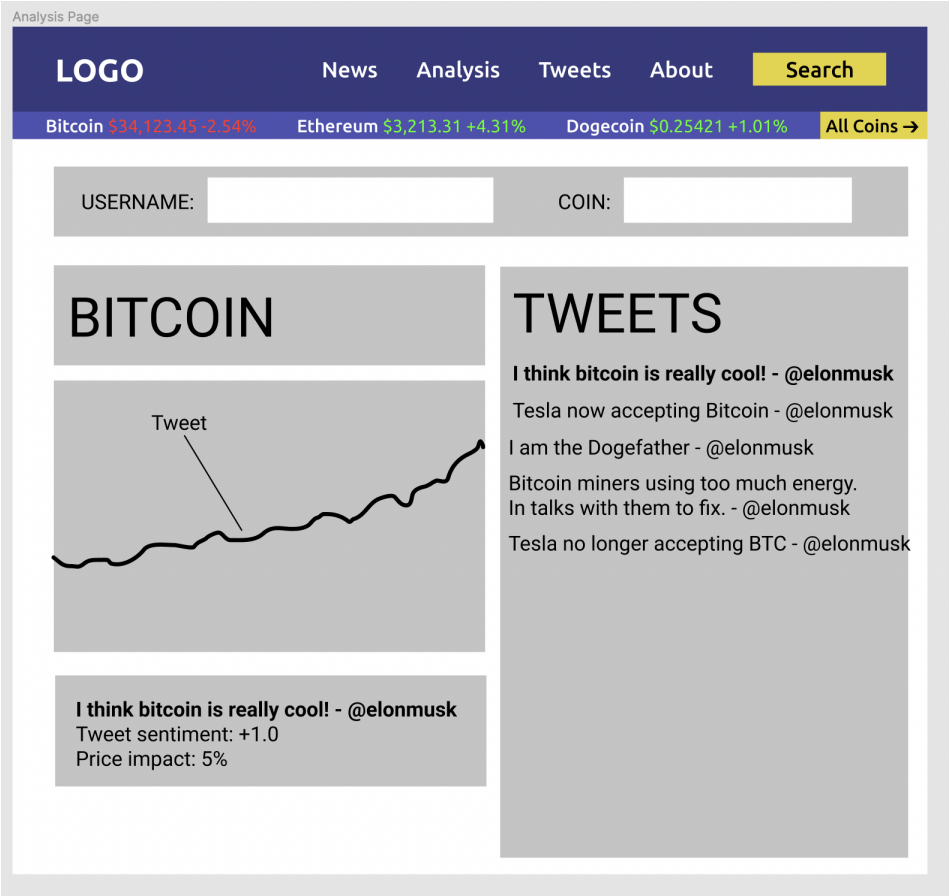

Figure 17: Analysis Page Mockup

Finally, I made a mockup of what the news page could look like. This page should be a scrollable list of all the articles sorted by most recently added. At the top there could potentially be a full width 'Featured' article, which could either be specifically chosen or the most recent article. Clicking on any of the articles should take you to the page for the article.

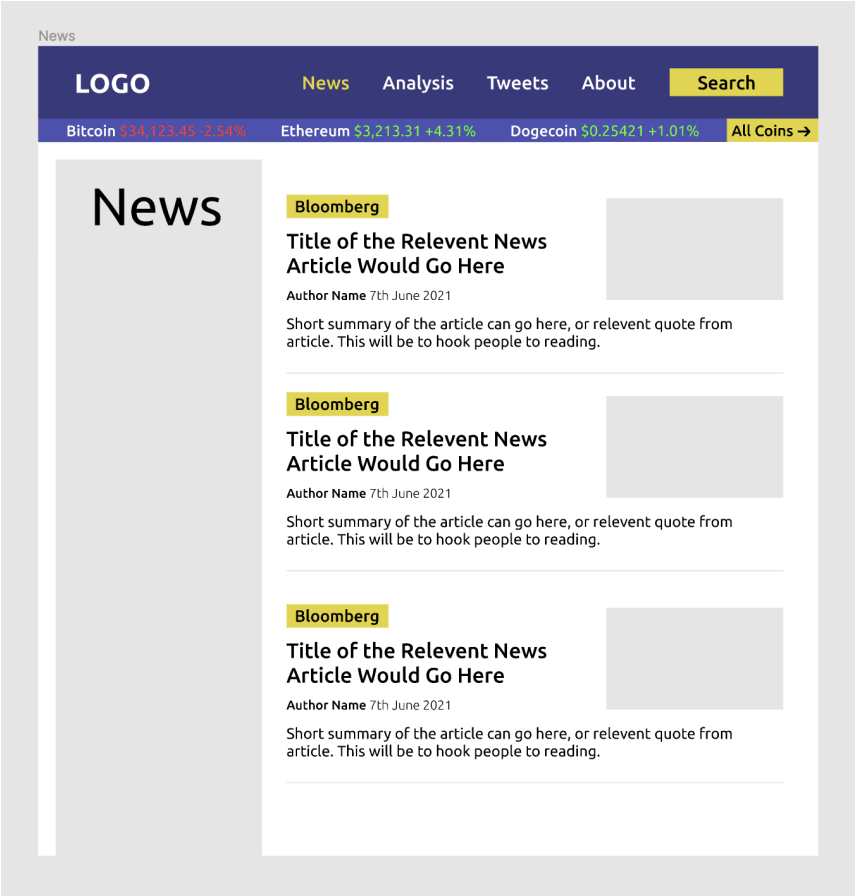

Figure 18: News Page Mockup

One thing to note with the mockups is that the design is one of the least important parts of this project. I will aim to complete all the functionality first, and then will work on the appearance after.

COMMON SECURITY VULNERABILITIES AND MITIGATION

I need to ensure that my application is secure against common security vulnerabilities, and I will need to take proper steps to ensure that my application is secure against the common ones. Below I have researched some of the common security vulnerabilities that I will be ensuring my program is robust about, and I have mentioned some common mitigation techniques against said vulnerabilities.

SQL Injection

SQL Injection is one of the most common web vulnerabilities, that involves submitting a malicious payload to the website that ends up being executed by the SQL server.

The following is an example of a function that would be vulnerable to SQL injection.

```
1 def fetch_users_password(username):
2     SQL_QUERY = "SELECT password FROM users_table WHERE username
         ='" + username + "';"
3     return execute_sql(SQL_QUERY)
```

If a hacker was to enter `' OR 1=1` into the following function, the SQL query would return a list of all the users password, rather than the specified user. This is because `1=1` always evaluates to True, and the OR statement means that the WHERE clause is true for all entries in the database, resulting in all entries being returned.

Mitigation To ensure my application is secure against SQL Injection attacks, I will be using prepared SQL statements. Prepared statements are a feature commonly provided in SQL libraries, that allows the user to provide parameters to an SQL query, rather than having to include the parameter values in the statement itself.

For example, the following statement does not use prepared statements: `SELECT * FROM users WHERE name = 'arthur'`; This could be remade using prepared SQL statements to look like the following: `SELECT * FROM users WHERE name = $1`; Then, in this example the value `arthur` could be supplied as a parameter. This prevents SQL injection attacks such as the `fetch_users_password` example above.

Cross Site Scripting

Cross-Site Scripting (XSS) is similar to SQL Injection, however typically involves malicious JavaScript code being injected into a webpage rather than SQL queries being abused. It typically happens as a result of an application not filtering and sanitising user input. For example, in a comment text field, I should not be able to insert HTML tags into the page.

Mitigation When handling user input, I will perform server side sanitation and validation to ensure that the user's input is not malicious. I will limit allowed characters, and perform

regex validation against text to ensure that there is no way an attack can inject code into a page. In addition, I will HTML encode any comments that are being displayed onto my site. HTML encoding turns characters such as < (which is used to open a HTML tag) into other symbols that do not impact the page. For example, the string `<script>` would be turned into `<script>`. When viewed on the page however, it will appear as the original string.

Broken Access Control

Broken Access Control vulnerabilities are when there is a lack of authorisation check when attempting to access privileged resources / areas of a website. For example, as a user of a bank I should be able to access my bank account balance, but not someone else's. According to OWASP.org, Broken Access Control is one of the most common website vulnerability seen.

Mitigation To mitigate against this sort of vulnerability, I will be creating functions to limit access to specific pages and API routes. When creating a new page or API route, I will consider who the intended user is, and carefully manage who can access. In addition when testing, I will ensure that no user can access resources that I know they shouldn't be able to.

SECURITY MEASURES

Below I have detailed the functionality behind some of the security measures I will be implementing. I have also discussed some possible extensions.

JSON Web Tokens and RSA

As mentioned in the analysis section, I will be using JSON Web Tokens as a method for authenticating and verifying my users identity.

JSON Web Tokens are an open standard (RFC 7519) for implementing a secure way to transmit information between two parties (in my case the client and server) as a JSON object. This information can be verified by making use of digital signatures. In my case, I will be signing my JWTs using an RSA private key that I will generate. JWTs can be signed by a variety of different algorithms, including ECDSA, and RSA. I chose to use an Asymmetric Key algorithm to sign my JSON Web Tokens, as I am familiar with the core concepts behind them. This left me with 2 main options, ECDSA or RSA. I made the following comparison table to help me choose:

	ECDSA	RSA
Type	Asymmetric Public/Private Key	Asymmetric Public/Private Key
Complexity	High Complexity	Simpler than ECDSA to implement
Key Length	Much shorter keys required to provide the same security	Typically uses 2048-bit or 4096-bit keys
Standardised Date	2005	1995
Widespread Use	Less adopted than RSA	Most widely used asymmetric algorithm
Core Concept	Works on the mathematical representation of Elliptical Curves	Works on the principle of the Prime Factorisation problem

RSA I ended up on choosing RSA. I have done some work with RSA before, so I am already familiar with how it mathematically works. In addition, it is still one of the most popular choices for encryption algorithms and has been used for over 25 years, proving it has stood the test of time.

RSA works on the prime factorisation problem. This put simply is the fact that two very large prime numbers multiplied together produce a semiprime number. It is easy mathematically to multiply the primes to form the semiprime, but it is incredibly difficult and computationally hard to factorise the semi prime back into its original two prime numbers. RSA works in the following way:

Generating Keys

1. You select two large prime numbers, p and q.
2. Calculate their product. $n = p \times q$
3. Calculate the totient function. $\phi(n) = (p-1)(q-1)$
4. Select a value of e. e should be coprime to $\phi(n)$ and $1 < e < \phi(n)$. Numbers are coprime if 1 is the only positive integer that divides them. In practice 65537 is very commonly used as e, because it is a Fermat prime and is of suitable size for security.

5. The Public Key is the pair of numbers n, e. This can be shared to any party.
6. The Private Key (d) is calculated from the numbers p, q, and e. The numbers are related with the Extended Euclidian Algorithm, which proves that $e \times d = 1 \, mod \, \phi(n)$. d can be found from this.
7. The Private Key is the pair of numbers n, d. This should be kept secret and is what will be used to encrypt messages.

Encryption The following is an equation to encrypt using the previously found values for the public key. P represents the plaintext, and C represent the cipher-text.

$C = P^e \, mod \, n$

Decryption Decrypting follows a very similar process, though this time it uses the private key.

$P = C^d \, mod \, n$

You can find some pseudocode and further information on RSA further down this document.

JWT

I chose to use JSON Web Tokens alongside RSA. JSON Web Tokens are composed of 3 parts separated by dots, which are the **Header**, **Payload**, and **Signature**. A typical JWT looks like the following:

```
xxxxx.yyyyy.zzzzz
```

Header The header contains information about the token, including the algorithm, and the type. In my case, I am using RS256. This means I am using RSA, with SHA256 as the hashing algorithm. My header will look like this:

```
{
    "alg": "RS256",
    "typ": "JWT"
}
```

This JSON is then encoded using Base64, to produce a string that looks like the following:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9

Payload The payload contains the data that we want to transfer and verify between two parties. This payload is then Base64 encoded, to produce another string. In my case, the payload will likely contain the user's email, their permissions, and the expiry time. The expiry time denotes how long the token should be considered valid for. Choosing an expiry time is a trade off between convenience and security - lower expiry means higher security, but requires the user to authenticate more often.

Signature Finally, there is the signature. The signature is the most important part, and is what ensures that we can trust the data in the payload is genuine and not modified. As I am using RSA, the signature can also be used by the client to verify that the JWT has originated from me using my public key.

The signature is made by first combining the header and payload with a '.' in between. Then, the RSA algorithm is applied to the result, using the hidden Private Key. This result is then hashed using SHA256.

Combining the header, payload, and signature with a '.' separating them produces our final token, which can be sent to the client after they've authenticated. This will then be stored in their cookies, and sent with all future requests to the server.

Verifying JWTs It is important that the JWTs are verified before the payload's contents are trusted. RSA and SHA256 produce the same outputs each time when the same inputs are supplied. Therefore, to verify that the signature is correct, we can use the Base64 encoded payload and header to recreate the signature with our Private Key. We can then compare our newly created signature with the signature provided by the client. If they match, then we have verified that the JWT is correct and was created by us, and we can trust it. And if they do not match, then we can assume the JWT has been tampered with and is untrustworthy.

Authentication Walls

My users will have to login to access key functionality. My login page will send the login form data to my API, where it will be checked to ensure that it's valid. In the event that it is, my server will return a signed JSON web token to the user, which will be stored in the browser's cookies with a short expiry. All subsequent requests to my API server will contain an authorisation header with the token, which ensures that my server can verify who any requests came from and that they are authorised.

Not all of my application and requests will be behind a login screen however. I will create a function that allows certain routes to be protected, and others not.

API Server Security

I will be storing password hashes in my PostgresQL database, rather than plaintext. I will be using the Argon2 hash function as previously described, which has excellent resistance to GPU cracking and is suitable for storing passwords. When verifying passwords are correct, I can make use of the verify function to compare a plaintext password with the hashed password from the database. I will also be salting the passwords, which mitigates against hash table attacks in the event of a data breach. These measures will help ensure my user's information remains secure. Salting is a practice involving adding a unique random string of characters known only to the site to each password before it is hashed. This salt value is typically stored in plaintext by the site, and is recalled when verifying a hash to ensure a password is correct. Salts are a safeguarding method that ensures that even if two users have the same password, the hash produced will be different. This can drastically slow down malicious hackers in the event of the password database being leaked. My chosen password algorithm Argon2 automatically handles the generation and use of salts.

In addition, I will implement a password security requirement when signing up to ensure that users use a secure password. The password requirements I will be enforcing are the following: 1. At least eight characters 2. At least one number 3. At least one uppercase letter 4. At least one special character To implement this, I will be using a regex rule to match passwords that meet the requirements. I will be using the following pattern: `^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[\W]).{8,}$` This complex pattern I developed checks against all my defined requirements.

Testing Phase

In my testing phase, I will also be checking that my application is secure against common website exploits, such as SQL and XSS injection which I have mentioned in the analysis section. These sort of injection attacks involve sending maliciously crafted payloads in fields that accept user input, such as login forms. Attackers hope that these payloads cause code to be executed on the website server, which if successful would allow a malicious party to gain complete access to the server. It is essential that no application is vulnerable to such attack. My testing plan will detail how I will attempt to verify that my application is secure.

Additional Possible Measures

There are many additional security measures that I could implement to further improve my applications security. It is unlikely that within the time period of this project I will be able to implement any of the following, however they all act as possible extensions for the future.

Rate Limiting IP Based Rate limiting would mean that if a user makes too many requests from one IP address in a short period of time, they would be temporarily blocked and prevented from making further requests. This is an important feature that stops attackers from brute forcing things such as password login attempts. Rate Limiting aims to only stop bots and automated programs from making too many requests too fast - it should never affect a user using the application typically. Ideally given time, this should be implemented either using a library or a custom program.

CAPTCHA and Anti-Robot buttons CAPTCHA stands for the Completely Automated Public Turing test to tell Computers and Humans Apart. They are designed to be able to distinguish between real humans and robots, by providing pop up challenges to complete when clicked. They are commonly used in login or registration forms to ensure that the person accessing the application is real. CAPTCHA's are very effective at stopping robots and automatic program from using an application. This can stop attacks such as a site being flooded with spam. CAPTCHA's can be implemented using services such as Google's reCAPTCHA. They are very important with any public application.

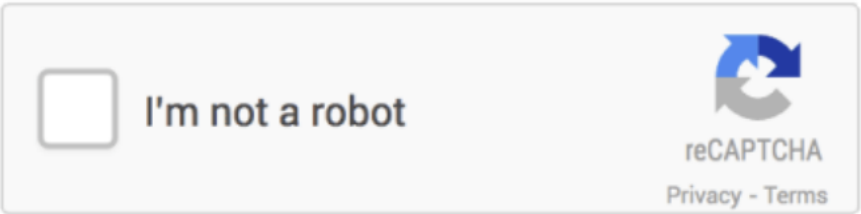

Figure 19: Screenshot of what a Captcha typically looks like

Multi-Factor Authentication Multi-Factor Authentication (MFA) is an authentication technique that requires a user to provide multiple verification factors to be able to confirm their

login. One of the most common MFA factors used are one-time passwords. One-time passwords are codes typically sent via SMS or email to a user, that needs to be entered to access the application. This adds an extra layer of protection, as it means that even if an attacker has your password, they require an extra layer to gain access to your account. Most MFA techniques are based on one of three things:

Knowledge These are things you know. These include security questions, additional PINs, or other passwords.

Inherence These include permanent attributes that you as a person have. For example, voice, fingerprint, iris or other biometric recognition.

Possession Possession includes things that you have access to and control of. For example, you have access to your smartphone which can receive a one-time password over SMS, or you have access to your email. You can also get security USB keys, which requires the physical device present to authenticate.

Most important apps will require you to have some form of Multi-Factor authentication these days, especially apps that relate to finance.

IP Tracking and Blocking Many secure applications use your IP address and other device information when you are authenticating as an extra layer of security. They aim to identify suspicious patterns and halt them. For example, if you consistently log into a website from the UK, and then suddenly attempt to login from Russia, the login attempt might be flagged as suspicious and stopped. This is tricky to implement, and often requires the use of machine learning models to attempt to predict users behavior.

BACKUPS

Frequent backups are important with all online applications. The source code for the program will be backed up onto GitHub. GitHub is a version control system that acts as a code repository and tracks changes over time. GitHub is a free cloud service used by millions, meaning that even if the source code is accidentally deleted from the computer while developing, it is secure in the cloud and can be easily restored.

When the program is running, the Postgres database will need to be backed up as well. Postgres offers 3 main approaches to backing up data:

SQL Dump This method involves creating a "dump" of the database. It will generate a text file of SQL commands that can be run again on the server to recreate the database to the same state as when the dump was created. PostgreSQL offers a built in way of doing this, through the `pgdump` command. Running this command produces a set of SQL commands that should be saved to a file for later use. Restoring is as simple as pasting the commands back into the SQL shell. I will setup an automatic system to before this type of backup weekly, and the backup dumps will be saved to an external server.

File System Level Backups Another method that PostgreSQL offers for backups are File System Level Backups. These involve directly copying the files PostgreSQL generates to store the data in a database, and restoring them at a later point when required. This method is however not as suited as SQL Dumps, as the database server must be shut down in order to get a backup. In addition, the file size generated by this backup is typically much larger than an SQL dump. This method also requires advanced knowledge of the UNIX file system - restoring an SQL dump is a much easier experience for the client.

Continuous Archiving and Point-in-Time Recovery Continuous Archiving and Point-in-Time Recovery is PostgreSQL's equivalent to an incremental backup. Incremental backups are a backup of all changes made since the last backup. With incremental backups, there is normally one full backup done first. Then, future backups just track changes since then. This helps save in storage size and normally results in much faster backups. If I was expecting the database to grow to a large size, I would be using this backup method. However, given the limited nature of what is being stored I do not expect the database to grow past a few megabytes in size. This makes SQL dumps more suited, due to the low complexity required to setup, compared to the advanced setup required with an incremental backup.

SENTIMENT ANALYSIS

The sentiment analysis portion of the project will likely be a technically complex part. As per my clients request, I will be making a model to analyse tweets and the sentiment of their content. You can view some details about how I will be doing this below.

Algorithm

I have chosen to use a bidirectional Long Short-Term Memory neural network. LSTMs are a type of Recurrent Neural Networks capable of processing entire sequences of data. They are particularly suited to classifying text, which is why I will be using one. This will be implemented with the use of TensorFlow and Keras, which have built in support.

Dataset

To train my machine learning model I will be using a dataset I have found on the website Kaggle. Kaggle is a Google owned company that allows users to publish and find data sets for purposes such as my own. The dataset I have chosen to use is a collection of tweets with their sentiment already classified. My machine learning model will then use this dataset to learn from and train itself. The dataset comes from the following link: https://www.kaggle.com/c/tweet-sentiment-extraction/data

The dataset consists of a CSV file with over 27,000 rows. Each row contains 4 columns, `textID`, `text`, `sentiment`, and `selected_text`. `textID` is a unique ID for each row, `text` is the original tweet, `sentiment` is either `neutral`, `positive`, or `negative` depending on the content, and `selected_text` is the part of the text that is responsible for the sentiment.

For example, the following is a sample from the dataset:

textID	text	sentiment	selected_text
997a62f83f	These kids are terrible! If I was in Good Evans, I'd call Childline	negative	These kids are terrible!

For my purposes, I am just interested in the `sentiment` and `selected_text` column, which will provide enough information to train a model.

This dataset will need preprocessing. Preprocessing is the act of removing unwanted parts and turning the dataset into something useful to a computer. In my case, this will include removing punctuation, URLS, emails, and other unwanted characters from the dataset. I will create a function to do this.

Before the dataset can be interpreted, it needs to be Tokenized. Tokenization is the process of splitting up each text into smaller pieces such as individual words or phrases, called Tokens. Algorithms typically need text to be tokenised to understand what is going on.

The dataset will also be split into two parts, a training set, and a testing set. The training set will consist of 75% of the dataset. This will be used to create the model. The other 25% will consist of a testing set. This will be used after the model is created to test and evaluate it's accuracy.

Training

Training a neural network is hardware intensive. For this reason, I will be making use of the free service Google Collab. Google Collab offers free access to powerful GPUs and lets users run python programs in the cloud. I expect the program to take several hours to complete training.

Exporting

Once the model is trained, I will export it as a Pickled object. Pickling is the process of converting a Python object into a byte stream that can be stored. I will download the Pickled trained model from Google Collab for use in my API server. Then it will be as simple as developing a function to unpickle the file, allowing access to the trained model. Then, the API will have access to all the models functionality, and can be used to detect sentiment.

SERVER HARDWARE

My web application will need several components constantly running to ensure 24/7 uptime.

Client Frontend

I will be running my frontend client application using an online service called Vercel. Vercel is a service made by the creators of Next.js, the JavaScript library my frontend is using, and is suited towards hosting Next.js apps. Vercel offers a generous free tier that will allow my app to be hosted on their network of cloud servers at no cost. I will then be able to create a DNS

record on a domain of my choice to point towards Vercel's servers, allowing easy access to the frontend application.

API Server and Database

My API server and Database will use another online free service called Heroku. Heroku is capable of hosting Python web applications such as the API server, and can also create and manage databases. Heroku manages maintenance of the server and database, allowing for an easy development and maintenance experience.

ALGORITHM DESIGN

Sentiment Analysis

I will be using the Python Module Tensorflow to train my sentiment analysis model. Tensorflow abstracts away from much of the underlying code, however it will still require me to configure and train a model.

The dataset I am using does not come preprocessed, and contains raw tweets. This means the text in the dataset contains unwanted features, such as hashtags, URLs, and emojis. These are not useful to train a sentiment analysis model on, as I am just interested in the text meaning instead. For this reason, I will need to preprocess my dataset and turn it into a friendly format. I have described some of the preprocessing algorithms I will need to develop below:

Load_Dataset Function This function will use a built in CSV module to load a dataset from a `.csv` file, and return the relevant columns to be stored as a variable.

```
FUNCTION LOAD_DATASET(Path)
    Dataset = LOAD_CSV_FILE(Path)
    Dataset = Dataset[['selected_text', 'sentiment']]
    RETURN Dataset
ENDFUNCTION
```

Clean Function This function will take a dataset as its input. It will then perform Regex matching onto each item in the dataset and remove Regex matches from each item. It will then return a cleaned list. I have provided a description of what each Regex pattern does below.

This function also makes use of lambda, or “Anonymous” functions. Lambda functions are suited to single use functions that take use of other functions - in my case a regex substitution function. `Regex.SUB` takes three inputs. The first input is a regex rule to match against. The second input is a string to replace any found matches with. The third and final input is the string to test the regex against.

```
IMPORT Regex

FUNCTION CLEAN(Data)
    Data = Data.apply(lambda x: Regex.SUB(r'http\S+', '', x))
    Data = Data.apply(lambda x: Regex.SUB(r'#\S+', '', x))
    Data = Data.apply(lambda x: Regex.SUB(r'@\S+', '', x))
    Data = Data.apply(lambda x: Regex.SUB(r'[^\w\s]', '', x))
    Data = Data.apply(lambda x: Regex.SUB(r'\s+', ' ', x))
    Data = Data.apply(lambda x: Regex.SUb(r"\'", "", x))
    RETURN Data
ENDFUNCTION
```

Expression	Description
`$http\S+`	Matches all URLs
`#\S+`	Matches all #Hashtags
`@\S+`	Matches all @Mentions
`[^\w\s]`	Matches all non alphanumerical characters
`\s+`	Matches multiple sequential spaces
`\'"`	Matches single quotation marks

Create_Sequences Function I will be using an external Python module to create a tokenizer to apply to my dataset.

```
IMPORT Tokenizer

FUNCTION CREATE_SEQUENCES(Data)
    Tokenizer = Tokenizer()
    Tokenizer.FIT_ON_TEXTS(Data)
    RETURN Tokenizer.Texts_To_Sequences(Data)
ENDFUNCTION
```

Once I have my dataset preprocessed, training is fairly straightforward. You can see below some Pseudocode showing what my training file might look like.

```
IMPORT Tensorflow


Dataset = SHUFFLE(Dataset)

TRAINX = Dataset['selected_test'][:int(len(Dataset)*0.8)]
TESTX = Dataset['selected_test'][int(len(Dataset)*0.8):]
TRAINY = Dataset['sentiment'][:int(len(Dataset)*0.8)]
TESTY = Dataset['sentiment'][int(len(Dataset)*0.8):]

Model = LSTM()

Model.ADD(Layers.Embedding(MAX_WORDS=5000, INPUT_LENGTH=200))
Model.ADD(Layers.Bidirectional(Layers.LSTM(20, DROPOUT=0.6)))
Model.ADD(Layers.Dense(3, ACTIVATION='softmax'))

Model.COMPILE(OPTIMIZER='rmsprop', LOSS='
   categorical_crossentropy', METRICS=['accuracy'])
Model.FIT(TRAINX, TRAINY, EPOCHS=100)

OUTPUT(Model.EVALUATE(TESTX, TESTY))

TEXT_TO_TEST = INPUT('Enter a text to test: ')
OUTPUT(Model.Predict(TEXT_TO_TEST))
```

Authentication

I have made a flowchart demonstrating how authentication will be handled.

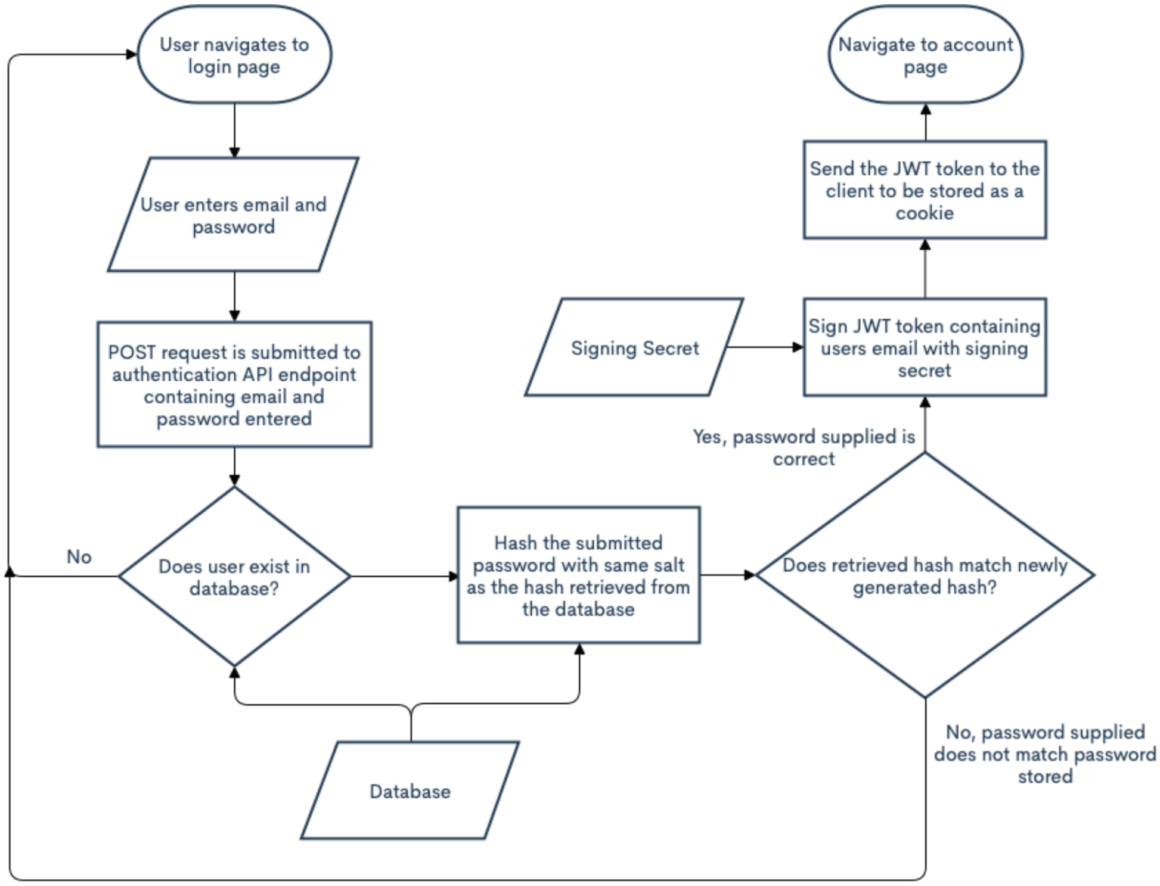

Figure 20: Flowchart of Authentication Flow

The authentication process will make use of several different functions and components of my project. I have created some Psuedocode for some of the key ones that will be used.

Create_JWT Function The Create_JWT function is a core part of the authentication flow. It makes use of the base64 functions that are described further down, and also makes use of the RSA keys created. It takes a payload as an input, which will contain data such as the user's email in JSON format.

```
FUNCTION CREATE_JWT(Payload, Private_Key):
 Header = '{"alg":"RS256","typ":"JWT"}'
 Header_Encoded = Base64.ENCODE(header.encode('utf-8')).decode('ascii').strip('=')
 Payload_Encoded = Base64.ENCODE(payload.encode('utf-8')).decode('ascii').strip('=')
```

```
 Body = Header_Encoded + '.' + Payload_Encoded

 Signature = RSA_SIGN(PRIVATE_KEY, Body, 'sha256')
 Signature_Base64 = Base64.ENCODE(Signature).STRIP('=').REPLACE('+', '-').REPLACE('/', '_')

 JWT = Body + '.' + Signature_Base64
 RETURN JWT
ENDFUNCTION
```

Verify_JWT Function The Verify_JWT function will be used to check a JWTs authenticity, by confirming that the signature is correct and signed by the private RSA key. For this, we can use the corresponding public key to check - a property of asymmetric cryptography.

```
FUNCTION VERIFY_JWT(JWT, Public_Key):
 Header, Body, Signature = JWT.SPLIT('.')
 Signature_Decoded = Base64.DECODE(Signature.REPLACE('-','+').REPLACE('_','/')+'==')

 TRY:
     RSA_VERIFY(Public_Key, Signature_Decoded, (Header + '..' + Body), 'sha256')
     RETURN True
 EXCEPT Exception: # Signature can't be verified
     return False
ENDFUNCTION
```

RSA (Rivest–Shamir–Adleman) Key Generator

RSA is an encryption algorithm that takes advantage of modular arithmetic principles. As previously described, I will be requiring some code to generate RSA keys. Below, I have detailed the different components and shown how they could be made. I have decided to split up the key generation into several different functions to improve readability.

Miller_Rabin Function The Miller Rabin primality test is an algorithm that attempts to estimate whether a number is likely to be a prime number. It is one of the simplest yet fastest tests known to solve this problem. RSA requires large prime numbers to be generated, so I

need a way of telling if a number is prime or not. I will be dealing with numbers up to the size `2**48`, so a lookup table of prime numbers would not be suitable.

The functions time complexity is the following: O$(k \log^3 n)$. k is how many rounds the function is to be performed. In my case, I have chosen to use 10 rounds. The round number is a trade of between performance and accuracy.

```
FUNCTION MILLER_RABIN(NUM)
    S = NUM - 1
    T = 0

    WHILE S MOD 2 == 0
        S = S // 2
        T = T + 1
    ENDWHILE

    FOR X IN RANGE(10) # repeat 10 times, for 10 rounds
        A = RANDOM.RANDINT(2, NUM - 1) # generate random number
            less than input num
        V = (A ** S) MOD NUM
        IF V NOT == 1
            I = 0
            WHILE V NOT == NUM - 1
                IF I == T
                    RETURN FALSE
                ELSE
                    I = I + 1
                    V = (V**2) MOD NUM
                ENDIF
            ENDWHILE
        ENDIF
    ENDFOR

    RETURN TRUE
```

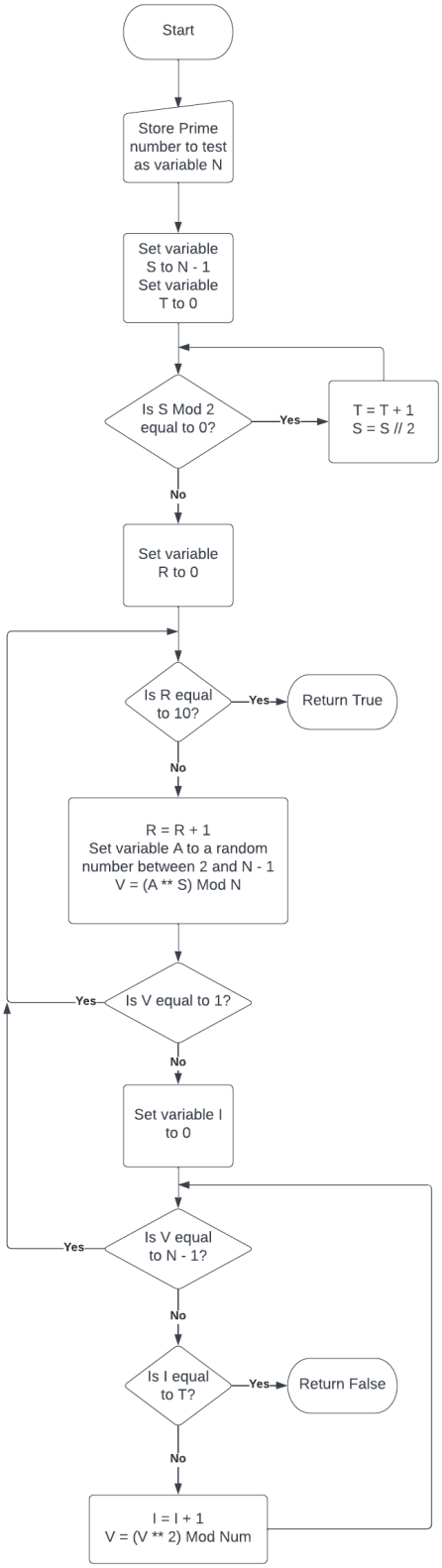

Figure 21: Miller Rabin Function Flowchart

Generate_Prime function This function will repeatedly generate a large number that has the specified keysize number of bits. It then estimates if the number is a prime number or not using the RABBIN_MILLER function. If it estimates the number to be prime, it returns the number. If not, it will repeat the process until it finds a prime.

```
KEYSIZE = 1024

FUNCTION GENERATE_PRIME(KEYSIZE)
    WHILE True
        NUM = RANDOM_INT(2**(KEYSIZE-1),2**KEYSIZE)) # generate a number of keysize bits
        IF MILLER_RABIN(NUM) == True
            RETURN NUM # keep on running loop until we generate a prime
        ENDIF
    ENDWHILE
ENDFUNCTION
```

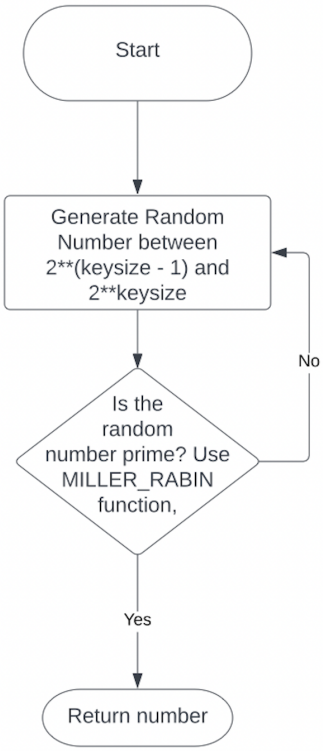

Figure 22: Generate Prime Function Flowchart

Extended_Euclidean_Algorithm Function The Extended Euclidean Algorithm is used during key generation to find the modular inverse of the value `E` with `(P - 1)* (Q - 1)`. Due to our previously defined functions to create the public and private key, we know that the value of `E` is relatively prime to `(P - 1)* (Q - 1)`. This means there exists integers `X` and `Y` such that `(E*X)+ ((P - 1)* (Q - 1)* Y)= 1`.

```
FUNCTION EGCD(A, B)
    IF A == 0: # in the case that A is 0, we need to return B, 0, 1
        RETURN (B, 0, 1)
    ELSE
        GCD, Y, X = EGCD(B % A, A) # recursively call the function, with the inputs B mod A, and A
        RETURN (GCD, X - (B // A) * Y, Y) # return a tuple using some of the output of the EGCD function. // is integer division.
    ENDIF
ENDFUNCTION
```

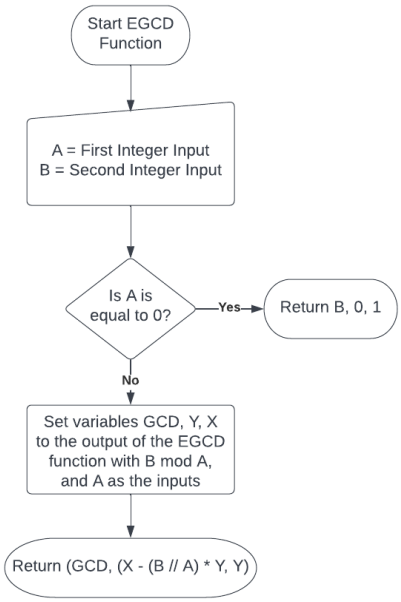

Figure 23: Euclidean Algorithm Flowchart

Generate_Key This will use the previously defined functions to create the final RSA keys for use. Each key consists of two number values. The Public key consists of the value for `N` with

the value `E`. The Private key consists of `N` and `D`. Further up in the document you can view an explanation of the mathematics behind RSA, and what each of the numbers signify.

```
P = GENERATE_PRIME(1024)
Q = GENERATE_PRIME(1024)

N = P * Q
E = 65537

G, X, Y = EGCD(E, (P - 1) * (Q - 1))

D = X % (P - 1) * (Q - 1)

PUBLICKEY = (N, E)
PRIVATEKEY = (N, D)
```

Base64

Base64 is a binary to text encoding scheme that can represent binary text in ASCII format. Base64 is typically used to encode data to be sent over a network.

Each Base64 digit represents 6 bits of data. I will be using Base64 as previously explained when I create my JSON Web Tokens to be sent to the client.

Add_zeros function This will be a recursive function used by both the encoding and decoding function, to add zeros padding to a binary value until the length of the binary is a multiple of 8. For example inputting `101011` will return `00101011`, `10001000` will return `10001000`, and inputting `1` will return `00000001`

```
BINARY = USERINPUT

FUNCTION ADD_ZEROS(BINARY)
    IF LEN(BINARY) MOD 8 NOT == 0 THEN
        BINARY = '0' + BINARY
        RETURN ADD_ZEROS(BINARY)
    ELSE
        RETURN BINARY
    ENDIF
ENDFUNCTION
```

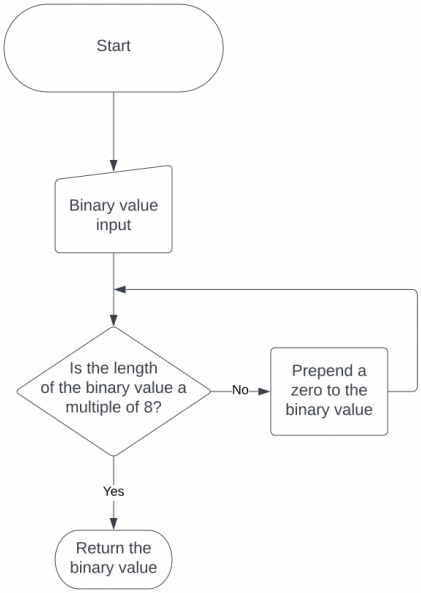

Figure 24: Add_Zeros Function Flowchart

Encoding This function will be used to encode text into base64.

```
constant TABLE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/' # defines the constant TABLE with all the base64 characters
BINARY = None # define varibale BINARY
TEXT = USERINPUT # defines the text variable, that assigned to the value of the user's input
BASE64 = None # define variable BASE64

FOR LETTER IN TEXT # iterate through each letter in the variable text
    BINARY <- BINARY + ADD_ZEROS(CODE_TO_BINARY(CHAR_TO_CODE(LETTER))) # append to binary, the binary representation of the letter with zeros padded
ENDFOR

WHILE LEN(BINARY) MOD 3 NOT == 0 # while the length of binary is not a multiple of 3:
    BINARY = BINARY + '00000000' # append 8 zeros. these are the padding characters
ENDWHILE
```

```

FOR NUM = 1 TO LEN(BINARY) # iterate through the length of the binary
    IF NUM MOD 6 == 0 THEN
        BINARY = BINARY[:NUM] + ' ' + BINARY[NUM:] # add a space every 6th digit
    ENDIF
ENDFOR

BINARY <- SPLIT(BINARY, ' ') # split the binary into a list at the spaces

FOR ITEM IN BINARY # iterate through the list of binary
    IF ITEM == '000000' # if the item in the list is 6 zeros
        BASE64 = BASE64 + '=' # then this is padding, and represented by an equals sign. add to the base64 string
    ELSE # otherwise
        BASE64 = BASE64 + TABLE[BINARY_TO_DECIMAL(ITEM)] # convert the item to decimal, then append the item in the table with the index of the decimal item to the base64 string
    ENDIF
ENDFOR

OUTPUT BASE64 # output the final result
```

Decoding Decoding follows a similar process, but in reverse. The following pseudocode details how this could be implemented.

```
constant TABLE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
BINARY = None
TEXT = None
BASE64 = USERINPUT

FOR LETTER IN BASE64
    IF LETTER == '='
        BINARY = BINARY + '000000'
    ELSE
        BINARY = BINARY + ADD_ZEROS(DECIMAL_TO_BINARY(INDEX(TABLE, LETTER)))
```

```
    ENDIF
ENDFOR

WHILE LEN(BINARY) MOD 3 NOT == 0
    BINARY = BINARY + '00000000'
ENDWHILE

BINARY = SPLIT(BINARY, ' ')

FOR ITEM IN BINARY
    IF NOT ITEM == '00000000'
        TEXT = TEXT + CODE_TO_CHAR(BINARY_TO_CODE(ITEM))

OUTPUT TEXT
```

TEST PLAN

To ensure my program is functioning as intended, I will need to carry out tests on all parts of my program.

I will test the accessibility and usability of my front end website's GUI. This will include using Google's Lighthouse website testing tool. Lighthouse runs a series of audits against the page, and then generates a report based on how well the page did. It is accessible from Chrome Dev Tools. The report ends up giving a score between 0 and 100, with a detailed breakdown. I would like to aim for a score of at least 90.

I will then do some black box testing. This will let me imitate a user, and will let me ensure that from a user's point of view, everything functions as intended. I can do this by interacting with my program and ensuring that it performs as expected without any faults. Here is where I will be testing that my program performs as expected, and produces correct graphs. I will create a checklist of functionality to test for. I will be recording a sample of this process and uploading it as a video.

I will be making use of tools to perform in-depth security testing on my application. Burp Suite is an integrated platform and graphical tool for performing security testing, which offers a free community license. I will be making use of this and previous experience to audit and test my application. Burp Suite offers many different features, the following of which I will be using:

- **Scanner** - This can be used to perform automatic scans of an application and flag any found vulnerabilities
- **Intercepter** - By proxying my network traffic through Burp Suite, I can view, intercept, and modify HTTP requests sent to my application in real time. This will allow me to test erroneous data and imitate an attacker.
- **Repeater** - Burp Suite automatically logs all requests made to an application, and allows them to be viewed retrospectively. It also allows these requests to be repeated, but with modified paramaters. I will use this to test for vulnerabilities such as XSS and SQL injection. I will ensure that all API routes are tested using this method, with a range of data supplied.

Finally, I will verify that several of my algorithms such as RSA function as intended by manually checking the mathematics and inputs. I will test that encryption and decryption successfully works using my RSA keys by using the formula I have described in the design phase. This will allow me to prove that my RSA keys are valid.

I will use the a table similar to the following to record my tests, and I will provide evidence as I go.

Description	Actual Result	Expected Result

I have completed a table full of the exact tests I will be carrying out in each component of the project. To save unnecassarily repeating, you can find the table with evidence in the testing stage of this document.

IMPLEMENTATION

TABLE OF FILES

My application is split up into three separate parts: Client, API, and Server.

The Client is the front end web application that the user interacts with. It will be the only part that the user will have to access, and provides a nice interface for accessing the applications functionality.

The API is another web application, but one that the user will not be required to directly use. It will be interacted with through HTTP requests, such as POST, GET, PUT, and DELETE. The Client will make these requests to the API on behalf of the user, and the API will return data that will then be interacted with by the client. The API will handle functionality such as authentication, data fetching and processing, and more.

The Server part is a collection of files that typically need to be run only once, or at specific times. Whilst the API and Client will need to be constantly running, the Server files will not. This will include files to do things such as updating the news database, generating RSA keys, and more.

A reasonable proportion of the files in the Client section of the application consists mainly of just HTML and CSS. For that reason, part of the Client section will not be annoted, unless there is any noteworthy algorithms in them.

The table below shows the list of files that my program contains, as well as a short description of their purpose and a reference to the page that their code is on.

File Path	Purpose
api/.env	Provide the environmental variables for the API section. Used for storing secrets.
api/main.py	The main file that launches and creates the API server.
api/api/auth.py	Provides the API routes to handle users logging in and registering accounts.
api/api/crypto.py	Provides an API route to return cryptocurrency price data from the Binance API.
api/api/news.py	Provides API routes to create, read, update, and delete news articles and comments from the backend database.

File Path	Purpose
api/api/twitter.py	Provides an API route to return a collection of tweets from Twitter's API for a specified user.
api/api/users.py	Provides API routes to query and access user information. Also provides CRUD routes to modify users.
api/core/auth.py	Provides functions used by the API routes to handle authentication.
api/core/binance.py	Contains a class for interacting with Binance's API to access cryptocurrency data.
api/core/security.py	Provides a class and functions for creating and verifying JSON Web Tokens, used in authentication.
api/db/crud.py	Provides a set of functions for interacting with the database using SQL queries.
api/db/schemas.py	Contains a set of classes representing the database models. Used by the FastAPI python module for interacting with API routes.
api/db/session.py	Contains a Session class for connecting to the database.
api/utils/base64.py	Contains a class for encoding and decoding between base64 and ascii.
client/.env	Provide the environmental variables for the Client section. Used for storing secrets.
client/next.config.js	Configuration file for Next.js
client/package.json	Configuration file for JavaScript
client/component/comments.js	HTML Components
client/component/loading.js	Loading Wheel Component
client/component/account/welcome.js	Welcome Banner Component

File Path	Purpose
client/component/admin/heatmap.js	Heatmap graph Component
client/component/admin/linechart.js	Linechart graph Component
client/component/admin/piechart.js	Piechart graph Component
client/component/admin/profile.js	Twitter Profile Component
client/component/admin/table.js	Table Component
client/component/admin/tableitem.js	Table Item Component
client/component/analysis/input.js	Text Input Component
client/component/analysis/ohcl.js	Candlestick Chart Component
client/component/analysis/search.js	Search Field Component
client/component/analysis/tweet.js	Tweet Component
client/component/analysis/user.js	Twitter User Component
client/component/coin/graph.js	Candlestick Chart Component
client/component/coin/relatednews.js	Related News Component
client/component/coin/sidearticle.js	Sidebar Article Component
client/component/coin/tableitem.js	Table Item Component
client/component/layout/layout.js	General Layout Component
client/component/layout/pagination.js	Pagination Support Component
client/component/layout/sidebar.js	Sidebar Component
client/component/layout/navbar/account.js	Navbar Button Component
client/component/layout/navbar/header.js	Navbar Header Component
client/component/layout/navbar/navbar.js	Navbar Component
client/component/layout/navbar/price.js	Navbar Price Subbar Component
client/component/layout/navbar/ticker.js	Navbar Ticker Component
client/component/layout/ticker/general.js	Navbar Ticker Component
client/component/layout/ticker/index.js	Navbar Ticker Component

File Path	Purpose
client/component/news/comments.js	Comments Field Component
client/component/news/content.js	News Content Component
client/component/news/feature.js	Feature News Article Component
client/component/news/post.js	News Post Component
client/services/auth.js	Authentication Service. A class for checking the status of a user's authentication, as well as performing authorised HTTP requests.
client/pages/account/index.js	Account Page
client/pages/account-analysis/index.js	Account Analysis Page
client/pages/admin/index.js	Admin Page
client/pages/tweet-analysis/index.js	Tweet Analysis Page
client/pages/coin/[coin].js	Specific Coin Page
client/pages/coin/index.js	Coin Leaderboard Index Page
client/pages/login/index.js	Login Page
client/pages/news/[id].js	News Article Page
client/pages/news/index.js	News Index Page
client/pages/register/index.js	Register Page
client/pages/users/[id].js	User Profile Page
client/pages/_app.js	Encases all pages in JS component
client/pages/_document.js	Adds HTML metadata to all pages
client/pages/404.js	Page displayed when client attempts to visit page that does not exist (404).
client/pages/global.css	Global CSS applied to every page.
server/news/fetch.py	Class that contains methods for fetching data from the News API, and interacting with the Database.

File Path	Purpose
server/news/db.py	Class that contains methods for interacting with the Postgres Database.
server/twitter/tweets.ipynb	Function to demonstrate and test fetching tweets and performing analysis using MatPlotLib and Tweepy.
server/rsa/keygen.py	Function to generate RSA Private and Public Key.
server/sentiment/train.py	File to train and create the neural network model

ADVANCED TECHNIQUES

I have provided a table that highlights some of the advanced programming techniques used, and which file you will find them in.

Technique	File(s)
Recursion	api/utils/base64.py, server/rsa/keygen.py
Advanced SQL Queries	api/db/crud.py, server/news/fetch.py
Complex Mathematical Algorithms	server/rsa/keygen.py
Object Orientated Programming Techniques	client/services/auth.js, api/core/security.py
Hashing and Encryption	api/core/security.py
Exception Handling	api/core/auth.py, api/core/security.py
Complex Client-Server Model	client/services/auth.js, client/pages/login/index.js, client/pages/tweet-analysis/index.js

Technique	File(s)
Parsing External Web Services APIs	client/pages/coin/[coin].js, client/services/auth.js, client/pages/account-analysis/index.js

ANNOTATED PROGRAM FILES

api/main.py

This file is responsible for launching the API server, and importing the different elements. It also connects the server to the database, and makes use of middleware. The middleware intercepts every request made to the server, and modifies the request object to include the database class instance. This means that the other files can access the same database class instance by accessing the request object. I did this for performance - rather than connecting to the database each time it needs to be queried, I can maintain a consistence connection that exists for as long as the file is running. This increases performance, and decreases server load to my database.

```
# third party module imports
from fastapi import FastAPI, Depends, Request
import uvicorn
from fastapi.middleware.cors import CORSMiddleware

# imports for the different endpoints/routes
from api.users import users_router
from api.auth import auth_router
from api.news import news_router
from api.crypto import crypto_api
from api.twitter import twitter_router

# imports for the database
from db.session import Session
from db.crud import Crud
# imports for authentication
from core.auth import get_user, get_admin

origins = [
    "http://localhost:5000",
    "http://localhost:8080",
```

```
    "http://localhost:3000",
    "https://nea.vercel.app",
    "https://cryptica.arthurr.co.uk",
    "*"
] # defines a list of origins for CORS requests

app = FastAPI(openapi_url=None) # creates the FastAPI app with no OpenAPI documentation
db = Crud("DATABASE_URL") # creates a database session using the DATABASE_URL environment variable

app.add_middleware( # adds the CORS middleware to the app. This allows cross-origin requests to be made from the defined list of origins
    CORSMiddleware,
    allow_origins=origins,
    allow_methods=["*"], # allows all HTTP methods, e.g. GET, POST, PUT, DELETE
    allow_headers=["*"], # allows all HTTP headers, e.g. Authorization, Content-Type
)
#

@app.middleware("http") # adds the middleware to the app. This allows the database session to be passed to the request object
async def db_session_middleware(request, call_next): # defines the middleware function
    request.state.db = db # adds the database to request.state.db
    response = await call_next(request) # calls the next middleware function
    return response # returns the response from the next middleware function

@app.on_event("startup") # adds the startup event to the app
async def startup():
    await db.connect()
    await db.create_table_users() # creates the users table if it doesn't exist
    await db.create_table_news() # creates the news table if it doesn't exist
    await db.create_table_comments() # creates the comments table if it doesn't exist

```

```
@app.on_event("shutdown") # adds the shutdown event to the app
async def shutdown():
    await db.close() # closes the database connection

@app.get("/api/hello") # defines the endpoint /api/hello
async def root():
    return {"message": "Hello World"} # returns Hello World. this is a test endpoint

app.include_router(users_router, prefix="/api/users",
                   dependencies=[Depends(get_user)]) # includes the users router in the app. This adds all the endpoints defined in the users file to the app. The prefix is the path that the endpoints will be added to. The dependencies are the dependencies that are required for the endpoints to be called.
app.include_router(auth_router, prefix="/api/auth") # includes the auth router in the app.
app.include_router(news_router, prefix="/api/news") # includes the news router in the app.
app.include_router(twitter_router, prefix="/api/twitter") # includes the twitter router in the app.
app.include_router(crypto_api, prefix="/api/crypto") # includes the crypto router in the app.

if __name__ == "__main__": # defines the main function that runs if the file is run directly
    uvicorn.run("main:app", host="0.0.0.0", reload=True, port=8000) # runs the app on the localhost on port 8000
```

api/api/auth.py

This file is responsible for the authentication routes of the API application. This contains a set of endpoints that allow the user to login, register, and verify their authentication status through HTTP requests. This file makes use of classes imported from other files, such as the `AccessToken` class which is responsible for the generation of JWT tokens.

```
from os import access
from fastapi import APIRouter, Depends, HTTPException, status, Request

```

```
4  from datetime import timedelta
5
6  from core import security
7  from core.auth import authenticate, sign_up, get_user
8  from db.schemas import UserLogin, UserCreate, UserEdit
9  import re
10
11 auth_router = router = APIRouter() # creates the auth router
12
13 @router.post("/login")
14 async def login(form_data: UserLogin, request: Request): # defines the login endpoint with the form data and the request object
15     user = await authenticate(request, form_data.email, form_data.password) # attempts to authenticates the user using the form data
16     if not user: # if the user doesn't exist)
17         raise HTTPException(
18             status_code=status.HTTP_401_UNAUTHORIZED,
19             detail="Incorrect username or password",
20             headers={"WWW-Authenticate": "Bearer"},
21         ) # returns a 401 error
22
23     if user['admin']: # if the user object has an admin field
24         permissions = "admin" # sets the permissions to admin
25     else: # otherwise
26         permissions = "user" # sets the permissions to user
27     access_token = security.AccessToken( # creates an access token object using the AccessToken class
28         data={"sub": user['email'], "permissions": permissions}) # sets the data to the user's email and permissions for the access token
29     return {"access_token": access_token.get_token(), "token_type": access_token.type} # returns the access token and the token type
30
31 @router.post("/register")
32 async def signup(form_data: UserCreate, request: Request): # defines the signup endpoint with the form data and the request object
33     signups_enabled = True # sets the signups_enabled variable to true. This can be changed to false to disable signups
34     if not signups_enabled: # if signups are disabled
35         raise HTTPException( # returns a 403 error
36             status_code=status.HTTP_403_FORBIDDEN,
```

```
            detail="Signups are currently disabled",
        )

    if not re.match(r'^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[\W]).{8,}$', form_data.password): # if the password doesn't match the regex
        raise HTTPException( # returns a 400 error
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Invalid Password",
        )
    if not re.match(r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$', form_data.email): # if the email doesn't match the regex
        raise HTTPException( # returns a 400 error
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Invalid Email",
        )
    if len(form_data.first_name) < 3 and len(form_data.last_name) < 3: # if the first name and last name are less than 3 characters
        raise HTTPException( # returns a 400 error
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Invalid Name",
        )

    user = await sign_up(request, # attempts to sign up the user using the form data
        form_data.email, form_data.password, form_data.first_name, form_data.last_name) # sets the user variable to the response of the sign_up function
    if not user: # if there is an error with the sign_up function and it does not return a created user
        raise HTTPException( # returns a 400 error as the user already exists
            status_code=status.HTTP_409_CONFLICT,
            detail="Account already exists",
            headers={"WWW-Authenticate": "Bearer"},
        )
    else: # otherwise
        access_token = security.AccessToken(  # creates an access token object using the AccessToken class
            data={"sub": user[0]['email'], "permissions": "user"}) # sets the data to the user's email and permissions for the access token
        return {"access_token": access_token.get_token(), "
```

```
                token_type": access_token.type} # returns the access
                token and the token type

@router.get("/me")
async def user_me(current_user=Depends(get_user)): # defines the
    /me that returns 200 if the user is authenticated
    if current_user['admin']: # if the current user is an admin
        return {'status': 200, 'admin': 'true', 'id':
            current_user['id']} # returns 200 and the admin field
             and the id field
    return {'status': 200} # otherwise returns 200

@router.put("/edit")
async def edit_user(form_data: UserEdit, current_user=Depends(
    get_user)):
    return
```

api/api/crypto.py

This file contains the API routes required to get cryptocurrency data from the API server. It uses the Binance class which is defined in another file to get historical data. The `get_price` function can take two inputs, one for the ticker (e.g. `BTCUSDT`), and one for the time in UNIX timestamp format. I have chosen to use UNIX to represent time as it can be represented as an integer, making it easy to work with and parse.

```
from core.auth import get_user, get_admin
import datetime
from core.binance import Binance
from fastapi import APIRouter, Depends, HTTPException, status,
    Request

crypto_api = router = APIRouter()
binance = Binance() # creates a new instance of the Binance
    class

@router.get("/{ticker}/{time}") # defines an endpoint that can
    take a ticker and a time parameter
def get_price(ticker, time, current_user=Depends(get_user)):
    start = int(time) - 1800 # sets the start variable to the
        time parameter minus 30 minutes
    stop = int(time) + 5400 # sets the stop variable to the time
        parameter plus 90 minutes
```

```
    return binance.get_historic(ticker, '1m', str(start), str(stop)) # returns the response of the get_historic function from the Binance class
```

api/api/news.py

This file contains the API routes for everything related to the News section of the site. This includes routes for fetching news, deleting news, searching for news, posting comments, and more.

```
import db.crud as crud
from core.auth import get_user, get_admin
import datetime
from fastapi import APIRouter, Depends, HTTPException, status, Request

news_router = router = APIRouter()

@router.get('/')
async def get_news(request: Request): # defines an endpoint that reutnrs all the news
    return await request.state.db.get_news() # returns the response of the get_news function from the crud class

@router.get('/comments/')
async def all_comments(request: Request, current_user=Depends(get_admin)): # defines an endpoint that returns all the comments. Requires admin permissions
    return await request.state.db.get_comments() # returns the response of the get_comments function from the crud class

@router.get('/{id}')
async def get_news_by_id(id, request: Request): # defines an endpoint that returns a news item by its id
    return await request.state.db.get_news_by_id(id) # returns the response of the get_news_by_id function from the crud class

@router.get('/{id}/comments')
async def get_news_comments(id, request: Request): # defines an endpoint that returns all the comments for a news item
    return await request.state.db.get_comments_by_news_id(id) # returns the response of the get_comments_by_news_id
```

```
        function from the crud class

@router.post('/{id}/comments')
async def create_comment(id: int, comment: dict, request: Request, current_user=Depends(get_user)): # defines an endpoint that creates a comment for a news item. Requires user to be authenticated and to provide a comment
    comments = {'user_id': current_user['id'], 'news_id': id,
                'date': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"), 'content': comment['content']} # creates a dictionary with the user id, news id, date and content of the comment
    return await request.state.db.create_comment(comments) # returns the response of the create_comment function from the crud class. Supplied with the comments dictionary

@router.delete('/{id}/comments/{comment_id}') # defines an endpoint that deletes a comment for a news item. Requires admin permissions or the user to be the author of the comment
async def delete_comment(id: int, comment_id: int, request: Request, current_user=Depends(get_user)): # supplied with the news id and comment id
    comment = await request.state.db.get_comments_by_id(comment_id) # returns the response of the get_comments_by_id function from the crud class
    if comment['user_id'] == current_user['id'] or current_user['admin']: # checks if the user is the author of the comment or an admin
        return await request.state.db.delete_comment(comment_id) # deletes the comment using the delete_comment function from the crud class
    else: # if the user is not the author of the comment or an admin
        raise HTTPException( # returns a 401 unauthorized error
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Unauthorized",
            headers={"WWW-Authenticate": "Bearer"},
        )

@router.post('/')
async def create_news(article: dict, request: Request, current_user=Depends(get_admin)): # defines an endpoint that creates a news item. Requires admin permissions
    return await request.state.db.create_news(article) # returns the response of the create_news function from the crud
```

```
        class. Supplied with the article dictionary

@router.post('/search')
async def search_phrase(search: dict, request: Request): # defines an endpoint that searches for a phrase in the news items
    return await request.state.db.get_news_by_phrase(search['phrase']) # returns the response of the get_news_by_phrase function from the crud class. Supplied with the phrase parameter
```

api/api/twitter.py

This file uses Twint, a python module for fetching tweets. It has a `/search` route that allows users to request tweets with a selection of search paramaters.

```
import random
import re
from types import SimpleNamespace
from fastapi import APIRouter, Request, Depends
import db.crud as crud
from core.auth import get_user, get_admin
import twint
import time
import nest_asyncio
from utils.sentiment import Sentiment
nest_asyncio.apply()
twitter_router = router = APIRouter()

twitter = twint.Config()
sentiment = Sentiment() # load the sentiment class

def get_user_details(username): # defines a function that gets user details from twitters API using twint
    u = twint.Config() # creates a new instance of the twint class
    u.Username = username # sets the username parameter to the username supplied
    u.Store_object = True # sets the store_object parameter to true
    twint.run.Lookup(u) # runs the Lookup function from the twint class
```

```
    return twint.output.users_list[-1] # returns the last item in the users_list variable from the twint class

@router.get('/search') # defines an endpoint that can take a query parameter
def get_tweets_phrase(coin=None, limit=100, minlikes=0, minretweets=0, minreplies=0, username=None, since=None, until=None, popular=None, retweets=None, sentiment=False, hide_tweets=None, current_user=Depends(get_user)):
    tweets = [] # creates an empty list to store the tweets
    if username: # if the username parameter is supplied
        username = re.sub(r'\W+', '', username) # removes all non a-Z characters from the username parameter
        user = get_user_details(username)  # gets the user details from the twint class
        twitter.Username = username.lower() # sets the username parameter to the username supplied
    if hide_tweets: # if the hide_tweets parameter is supplied
        return user # returns the user details
    if int(limit) > 1000: # if the limit parameter is greater than 1000
        limit = 1000 # cap the limit parameter to 1000
    twitter.Store_object = True # sets the store_object parameter to true
    twitter.Store_object_tweets_list = tweets # sets the store_object_tweets_list parameter to the tweets list
    twitter.Hide_output = True # sets the hide_output parameter to true
    twitter.Min_likes = int(minlikes) # sets the min_likes parameter to the minlikes parameter
    twitter.Min_replies = int(minreplies) # sets the min_replies parameter to the minreplies parameter
    twitter.Min_retweets = int(minretweets) # sets the min_retweets parameter to the minretweets parameter
    twitter.Search = coin # sets the search parameter to the coin parameter
    twitter.Limit = int(limit) # sets the limit parameter to the limit parameter
    twitter.Since = since # sets the since parameter to the since parameter
    twitter.Until = until # sets the until parameter to the until parameter
    twitter.Popular_tweets = bool(popular) # sets the popular_tweets parameter to the popular parameter
    twitter.Filter_retweets = bool(retweets) # sets the
```

```
filter_retweets parameter to the retweets parameter
    for a in range(1, 5): # loops through the range of 1 to 5. this is because twitter's API is buggy, and sometimes returns nothing even though there is data to be found. repeating the request 5 times if it returns nothing is a workaroud to this issue
        twint.run.Search(twitter) # runs the Search function from the twint class
        if len(tweets) > 0: # if the length of the tweets list is greater than 0
            break # break the loop
    if sentiment: # if the sentiment parameter is supplied
        for tweet in tweets: # loops through the tweets list
            tweet.sentiment = sentiment.predict(tweet.tweet) # sets the sentiment parameter of each tweet to the sentiment of the tweet
    if user: # if the user parameter is supplied
        return tweets, user # returns the tweets and the user details
    if len(tweets) == 0: # if the length of the tweets list is 0
        print("No tweets found for ", username) # print a message to the console
    return tweets # returns the tweets
```

api/api/users.py

This file contains the collection of API routes required to handle user authentication.

```
from db.schemas import UserCreate, UserEdit
from fastapi import APIRouter, Depends, Request

import db.crud as crud
from core.auth import get_user, get_admin

users_router = router = APIRouter()

@router.get("/admins")
async def admins(request: Request, current_user=Depends(get_admin)): # defines an endpoint that returns all the admins
    return await request.state.db.get_admins() # returns the response of the get_admins function from the crud class

```

```
@router.get("/count")
async def user_count(request: Request, current_user=Depends(get_admin)): # defines an endpoint that returns the number of users
    return await request.state.db.count_users() # returns the response of the count_users function from the crud class

@router.get("/",)
async def users(request: Request, current_user=Depends(get_admin)): # defines an endpoint that returns all the users
    return await request.state.db.get_users() # returns the response of the get_users function from the crud class

@router.get("/me")
async def user_me(request: Request, current_user=Depends(get_user)): # defines an endpoint that returns the current user
    return await request.state.db.get_profile( current_user['id'], True) # returns the response of the get_profile function from the crud class. Supplied with the user id and True for the profile parameter

@router.get("/{user_id}")
async def user_details(request: Request,
                       user_id: int,
                       current_user=Depends(get_admin)): # defines an endpoint that returns the user with the supplied id
    return await request.state.db.get_user( user_id) # returns the response of the get_user function from the crud class

@router.put("/me")
async def edit_profile(request: Request, data: UserEdit, current_user=Depends(get_user)): # defines an endpoint that edits the current user's profile
     return await request.state.db.edit_profile( current_user['id'], data) # returns the response of the edit_profile function from the crud class


@router.put("/{user_id}")
async def user_edit(request: Request, user_id: int, user: UserEdit, current_user=Depends(get_admin)): # defines an endpoint that edits the user with the supplied id
    return await request.state.db.edit_user( user_id, user) #
```

```
        returns the response of the edit_user function from the
        crud class

@router.delete("/{user_id}")
async def user_delete(request: Request,
                      user_id: int,
                      current_user=Depends(get_admin),
                      ): # defines an endpoint that deletes the
                         user with the supplied id

    return await request.state.db.delete_user( user_id) #
        returns the response of the delete_user function from the
         crud class

@router.post("/")
async def user_create(request: Request, user: UserCreate,
    current_user=Depends(get_admin),): # defines an endpoint that
     creates a new user. requires admin privileges
    return await request.state.db.create_user( user) # returns
        the response of the create_user function from the crud
        class

@router.get("/{user_id}/profile")
async def user_comments(request: Request, user_id: int): #
    defines an endpoint that returns the user's comments
    return await request.state.db.get_profile( user_id, False) #
         returns the response of the get_profile function from
        the crud class
```

api/core/auth.py

This file contains a collection of functions that are used throughout the API to verify and get user's details.

```
from fastapi import Depends, HTTPException, status, Request

from core import security

async def get_user(request: Request): # defines the get_user
    function. this function is used to get the current user from
    the request, which can then be used to verify the user's
    identity
```

```
    error = HTTPException( # defines the error variable as an HTTPException
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Failed to validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    token = request.headers['Authorization'].replace('Bearer ', '') # gets the token from the request headers
    try: # attempts to decode the token
        payload = security.AccessToken(token=token) # creates an access token object using the AccessToken class
        payload_data = payload.get_data() # gets the data from the payload
        email = payload_data['sub'] # gets the email from the payload
        if email is None: # if the email is None
            raise error # returns the error)
        permissions = payload_data['permissions'] # gets the permissions from the payload
    except Exception as e: # if the token is invalid and the RSA signature is invalid
        print(e)
        raise error # returns the error
    user = await request.state.db.get_user_by_email(email)
    if user is None: # if the user is None
        raise error # returns the error
    if user['admin'] == False and permissions == 'admin': # if the user is not an admin and the permissions in the token are admin
        raise error # returns the error
    return user # returns the user

async def get_admin( # defines the get_admin function. this is similar to the get_user function, but it only returns users with admin permissions
    current_user=Depends(get_user), # gets the current user from the request
):
    if not current_user['admin']: # if the current user is not an admin
        raise HTTPException( # returns an HTTPException
            status_code=403, detail="The user doesn't have enough privileges"
        )
```

```
    return current_user # returns the current user if the user is authenticated and the user is an admin

async def authenticate(request: Request, email, password): # defines the authenticate function. this function is used to authenticate the user
    user = await request.state.db.get_user_by_email(email) # gets the user from the database using the email
    if not user: # if the user is not found
        return False # returns false
    if not security.verify_password(password, user['hashed_password']): # if the password is not correct and the supplied password hash is not equal to the hashed password stored in the database
        return False # returns false
    return user  # returns the user if the password is correct

async def sign_up(request: Request, email, password, first_name, last_name): # defines the sign_up function. this function is used to sign up a new user
    user = await request.state.db.get_user_by_email(email) # gets the user from the database using the email
    if user: # if the user is found
        return False  # user already exists
    return await request.state.db.create_user( # else creates the user using the create_user function and a dictionary containing the user's supplied details
        {
            'email': email,
            'password': password,
            'first_name': first_name,
            'last_name': last_name,
            'admin': False}
    ),
```

api/core/binance.py

This file contains a class for interacting with Binance's API to fetch data related to Cryptocurrencies. It inherits from the third party python-binance module.

```
import os
from binance.client import Client as Client

```

```
class Binance(Client): # creates a new class called Binance that inherits from the Client class from the binance library
    def __init__(self):
        self.binance_key = os.getenv('BINANCE_KEY') # gets the BINANCE_KEY from the environment variables
        self.binance_secret = os.getenv('BINANCE_SECRET') # gets the BINANCE_SECRET from the environment variables
        super().__init__(self.binance_key, self.binance_secret) # calls the __init__ function from the binance library and passes the BINANCE_KEY and BINANCE_SECRET as parameters

    def get_historic(self, ticker, interval, start, stop): # creates a function called get_historic that takes a ticker, interval, start and stop as parameters
        return self.get_historical_klines(ticker, interval, start, stop) # returns the response of the get_historical_klines function from the binance library
```

api/core/security.py

This file contains the `AccessToken` and `Argon2` class. The `Argon2` class is used for hasing and verifying passwords using the argon2 module. The `AccessToken` class is used for generating and verifying JSON Web Tokens. The AccessToken class can be initialised with two methods. One, by supplying user data to be turned into a JSON Web Token. In this case, upon initilisation a JWT will be generated and accessable through the class. The other option is to intiliase the class by supplying it with an existing JSON Web Token. This can be used to verify that a token is valid, as if it is initialised this way, the class will attempt to decode and verify the tokens genuinity before allowing it to be accessed.

```
from passlib.hash import argon2 # from passlib.hash import the argon2 class
from datetime import datetime, timedelta # import the datetime and timedelta modules
import os # import the os module for loading environment variables
from utils.base64 import Base64 # import the base64 class from utils.base64
from OpenSSL import crypto # import the OpenSSL crypto module
from OpenSSL.crypto import X509 # import the X509 class from the
```

```
    OpenSSL crypto module
import json # import the json module for loading and dumping JSON

class Argon2:
    def __init__():
        return

    def hash_password(password): # define the hash_password function
        return argon2.hash(password) # return the argon2 hash of the password, using the argon2 class and an automatically generated salt

    def verify_password(password, password_hash): # define the verify_password function
        result = password_hash == argon2.using(salt=Base64.decode((
            password_hash.split(',')[2].split('$')[1] + '==').encode('utf-8'))).hash(password) # return the result of the password_hash being equal to the argon2 hash of the password, using the argon2 class and the salt from the password_hash
        return result # if they are equal it will return true, otherwise it will return false. If true the password is correct, if false the password is incorrect

base64 = Base64() # create a new instance of the Base64 class
class AccessToken(): # create Access Token Class
    def __init__(self, data=None, token=None): # initialises the class, allows the data and token to optionally be passed in
        self._private_key = os.getenv('RSA_PRIVATE_KEY') # gets the private key from the environment variables
        self._public_key = os.getenv('RSA_PUBLIC_KEY') # gets the public key from the environment variables
        self.algorithm = "RS256" # variable that sets the algorithm to RS256
        self.header = '{"alg":"'+self.algorithm+'","typ":"JWT"}' # variable that sets the header to the algorithm and type
        self.expires = 60 * 24 * 7 # sets the expiry of the token to 7 days (60 seconds * 24 hours * 7 days)
        self.type = 'bearer' # this is the name of the header in the web request that the token is sent in
```

```
        if data: # if the data is passed in
            self.data = data
            # add expiry to data. expiry is current timestamp + expiry
            self.data['exp'] = datetime.timestamp(
                datetime.now() + timedelta(minutes=self.expires))
            self.token = self.create_access_token() # create the token using the create_access_token function
        elif token: # if the token is passed in
            self.token = token # set the token to the token passed in
            if self.verify_token() != True: # check if the token is valid and the signature matches the public key
                raise Exception("Invalid token") # if not raise an exception
            self.data = self.__decode_token() # decode the token and set the data to the decoded token
            if "exp" not in self.data: # if the expiry is not in the data
                raise Exception("Token has no expiry") # raise an exception
            if self.data["exp"] < datetime.utcnow().timestamp(): # if the expiry is less than the current timestamp
                raise Exception("Token has expired") # raise an exception
        else: # if no data or token is passed in
            raise ValueError('You must provide either data or token') # raise an exception
        self.__init__()

    def get_token(self): # get the token
        return self.token

    def get_data(self): # get the data
        return self.data

    def __decode_token(self): # decode the token
        header, body, signature = self.token.split('.') # split the token into the header, body and signature at the .
        body_decoded = Base64.decode(
            body.replace('-', '+').replace('_', '/')+'==') #
```

```
decode the body with base64. Replace the - and _ with + and /
        return json.loads(body_decoded) # return the decoded body as a json object

    def verify_token(self): # verify the token
        header, body, signature = self.token.split('.') # split the token into the header, body and signature at the .
        signature_decoded = Base64.decode(
            signature.replace('-', '+').replace('_', '/')+'==') # decode the signature with base64. Replace the - and _ with + and /
        x509 = X509() # create a new x509 object. this is used to load the public key to then verify the signature
        x509.set_pubkey(crypto.load_publickey(
            crypto.FILETYPE_PEM, self._public_key)) # load the public key from the public key variable
        try: # try to verify the signature
            crypto.verify(x509, signature_decoded,
                          (header + '.' + body), 'sha256') # verify the signature against the header and body, using sha256 as the hashing algorithm and RSA")
            return True # if the signature is verified return true
        except Exception as e: # if the signature is not verified and an exception is raise return false")
            print(e)
            return False

    def create_access_token(self): # create the access token
        header_encoded = base64.encode(self.header).strip('=') # encode the header with base64. Strip the = from the end
        payload_encoded = base64.encode(
            str(self.data).replace("'", '"').replace(" ", "")).strip('=') # encode the payload with base64. Strip the = from the end
        body = header_encoded + '.' + payload_encoded # combine the header and payload
        pr_key = crypto.load_privatekey(crypto.FILETYPE_PEM, self._private_key) # load the private key from the private key variable
        signature = crypto.sign(pr_key, body, 'sha256') # sign
```

```
                the body with sha256 and the private key with RSA

        signature_base64 = Base64.encode(signature).decode(
            'ascii').strip('=').replace('+', '-').replace('/', '
                _') # encode the signature with base64. Strip the
                 = from the end. Replace the + and / with - and _
        jwt = body + '.' + signature_base64 # combine the body
            and signature
        return jwt # return the jwt

    def __str__(self): # string representation of the token
        return str(self.token)

    def __repr__(self): # representation of the token
        return str(self.token)
```

api/db/crud.py

This file contains a selection of classes for working with the database section of the application. The `Session` class is used for low level interactions with the database through the `asynpg` python module. Then, the `Table` class inherits from `Session` and provides some basic functions for working with general tables, such as a function to count how many rows in the database there are. Finally, there are three classes for each of the separate tables in my program. These contain functions specific to each of the tables, however they use the functionality inherited from the previous two classes. As mentioned in the

```
from core.security import hash_password
import asyncpg

class Session(): # creates a new class called Session
    def __init__(self, url):
        self.url = os.getenv(url), # gets the url from the
            environment variables
        self._cursor = None # creates a cursor variable
        self.con = None # creates a connection variable

    async def connect(self): # creates a function called connect
        self.connection_pool = await asyncpg.create_pool(dsn=
            self.url[0], max_size=3, min_size=1) # creates a
            connection pool using the url from the environment
            variables and the max_size and min_size parameters
```

```
        self.db = self.connection_pool

    async def close(self): # creates a function called close
        if self.connection_pool: # checks if the connection pool exists
            await self.connection_pool.close() # closes the connection pool

    async def execute(self, query, *args): # creates a function called execute
        async with self.connection_pool.acquire() as conn: # acquires a connection from the connection pool
            return await conn.execute(query, *args) # executes the query and returns the result

class Table(Session): # creates a new class called Table
    def __init__(self, url, table_name): # initializes the class
        super().__init__(url) # initializes the super class with the url passed in
        self.table_name = table_name # sets the table name variable to the table_name passed in

    async def drop_table(self): # creates a function to drop the table in the class
        await self.execute('DROP TABLE $1', self.table_name) # drops the table

    async def get_all(self): # creates a function to get all the data from the table
        query = f'SELECT * FROM {self.table_name}' # creates a query to get all the data from the table
        return await self.execute(query)

    async def get_by_column(self, column, id): # creates a function to get all the data from the table by a column
        query = f'SELECT * FROM {self.table_name} WHERE {column} = $1' # creates a query to get all the data from the table by a column
        return await self.execute(query, id)

    async def count(self): # creates a function to count the number of rows in the table
        query = f'SELECT COUNT(*) FROM {self.table_name}' # creates a query to count the number of rows in the table
```

```
        return await self.execute(query)

class News(Table): # creates a new class called News
    def __init__(self, url): # initializes the class
        super().__init__(url, 'news') # initializes the super class with the url passed in and the table_name passed in


    async def create_table_news(self): # creates a function to create the table in the class
        await self.execute( # creates a table called news
            'CREATE TABLE IF NOT EXISTS news (id serial PRIMARY KEY, publication varchar(255), author varchar(255), title varchar(511), description varchar(1023), url varchar(1023), imageUrl varchar(1023), date varchar(255), content varchar(1023))'
        )

    async def get_news_and_comments(self):
        news_and_comments = await self.fetch('SELECT * FROM news INNER JOIN comments ON news.id = comments.news_id ORDER BY date DESC') # creates a query to get all the data from the table, using inner join to get the comments
        if not news_and_comments: # checks if the news_and_comments variable is empty
            return None # returns none
        return news_and_comments # returns the news_and_comments variable

    async def get_news_and_comments_by_news_id(self, news_id):
        news_and_comments = await self.fetch('SELECT * FROM news INNER JOIN comments ON news.id = comments.news_id WHERE news.id = $1 ORDER BY date DESC', int(news_id)) # creates a query to get all the data from the table, using inner join to get the comments, sorting the comments by date
        if not news_and_comments: # checks if the news_and_comments variable is empty
            return None # returns none
        return news_and_comments # returns the news_and_comments variable


```

```
    async def get_news(self): # gets all news from the table sorted by date
        news = await self.fetch('SELECT id, title, publication, imageUrl, description, date FROM news ORDER BY date DESC')
        if not news:
            return None
        return news

    async def get_news_by_title(self, title): # gets all news from the table with matching title
        news = await self.fetch('SELECT * FROM news WHERE title = $1', title)
        if not news:
            return None
        return news

    async def get_news_by_author(self, author): # gets all news from the table with matching author
        news = await self.fetch('SELECT * FROM news WHERE author LIKE $1 ORDER BY date DESC', author)
        if not news:
            return None
        return news

    async def get_news_by_phrase(self, phrase, limit=5): # gets all news from the table with matching phrase in the title or description or content
        news = await self.fetch('SELECT title, imageurl, publication, id, date FROM news WHERE UPPER(title) LIKE $1 OR UPPER(description) LIKE $1 OR UPPER(content) LIKE $1 ORDER BY date DESC LIMIT $2', (f'%{phrase}%'.upper()), limit) # %{phrase}% is a wildcard to search for the phrase in the title or description or content
        if not news:
            return None
        return news

    async def get_news_by_id(self, id): # gets all news from the table with matching id
        if not id:
            return None
        news = await self.fetchrow('SELECT title, content, author, publication, imageurl, url, date FROM news
```

```
            WHERE id = $1', int(id)) # creates a query to get all the data from the table with matching id
        if not news:
            return None
        comments = await self.fetch('SELECT comments.id, user_id, content, date, first_name, last_name FROM comments INNER JOIN users ON comments.user_id = users.id WHERE news_id = $1 ORDER BY date DESC', int(id)) # query to get all the comments from the table with matching id using inner join to get the user
        if not comments:
            return dict(news) # return the news dictionary
        return {**dict(news), 'comments': comments} # returns the news dictionary with the comments dictionary

    async def create_news(self, news):
        await self.execute( # insert into news table with provided news dictionary
            'INSERT INTO news (publication, author, title, description, content, url, imageUrl, date) VALUES ($1, $2, $3, $4, $5, $6, $7, $8)',
            news['publication'], news['author'], news['title'], news['description'], news['content'], news['url'], news['imageUrl'], news['date']
        )

    async def delete_news(self, id):
        await self.execute( # delete from news table with provided id
            'DELETE FROM news WHERE id = $1', int(id)
        )
        return id

    async def edit_news(self, id, news):
        await self.execute('UPDATE news SET title = $1, description = $2, content = $3, author = $4, date = $5 WHERE id = $6',
                           news['title'], news['description'], news['content'], news['author'], news['date'], int(
                            id) # update news table by id with provided news dictionary
                           )
        return news

```

```python

class Comments(Table): # creates a new class called Comments
    def __init__(self, url):
        super().__init__(url, 'comments') # initializes the super class with the url passed in and the table_name passed in

    async def get_comments_by_id(self, id):
        comments = await self.fetchrow('SELECT * FROM comments WHERE id = $1', int(id)) # get comments from comments table with matching id
        if not comments:
            return None
        return dict(comments)



    async def create_table_comments(self):
        await self.execute( # create comments table
            'CREATE TABLE IF NOT EXISTS comments (id serial PRIMARY KEY, user_id int, news_id int, content varchar(2000), date varchar(255))'
        )

    async def get_comments(self):
        comments = await self.fetch('SELECT * FROM comments ORDER BY date DESC') # get all comments sorted by date
        if not comments:
            return None
        return comments

    async def get_comments_by_news_id(self, news_id):
        comments = await self.fetch('SELECT * FROM comments WHERE news_id = $1 ORDER BY date DESC', int(news_id)) # get comments matching a news article id
        if not comments:
            return None
        return comments

    async def create_comment(self, comment):
        await self.execute( # create comment in comments table using provided comment dictionary
            'INSERT INTO comments (user_id, news_id, content, date) VALUES ($1, $2, $3, $4)',
```

```
            comment['user_id'], comment['news_id'], comment['content'], comment['date']
        )
        return dict(comment)

    async def delete_comment(self, id):
        await self.execute( # delete comment from comments table with provided id
            'DELETE FROM comments WHERE id = $1', int(id)
        )
        return id

    async def edit_comment(self, id, comment):
        await self.execute('UPDATE comments SET user_id = $1, news_id = $2, content = $3, date = $4 WHERE id = $5',
                           comment['user_id'], comment['news_id'], comment['content'], comment['date'], int(
                            id) # update comments table by id with provided comment dictionary
                           )
        return comment
class Users(Table): # creates a new class called Users
    def __init__(self, url):
        super().__init__(url, 'users') # initializes the super class with the url passed in and the table_name passed in

    async def get_profile(self, user_id, me=False):
        if me: # if the user is the current user
            user = await self.fetchrow('SELECT id, first_name, last_name, email, admin FROM users WHERE id = $1', user_id) # get users info including email and admin status from the table
        else: # if the user is not the current user
            user = await self.fetchrow('SELECT id, first_name, last_name FROM users WHERE id = $1', user_id) # just get basic user info (name) from the table
        comments = await self.fetch('SELECT comments.id, news_id, comments.content, comments.date, news.title FROM comments INNER JOIN news ON comments.news_id = news.id WHERE user_id = $1 ORDER BY date DESC', int(user_id)) # fetch all comments from the table with matching user id
        if not comments:
```

```
            return user
        return {**user, 'comments': comments} # return the user dictionary with the comments dictionary



    async def get_user(self, id):
        user = await self.fetchrow('SELECT * FROM users WHERE id = $1', id) # select all users from the table with matching id
        if not user:
            return None
        return dict(user)

    async def get_user_by_first_name(self, first_name):
        user = await self.fetchrow('SELECT * FROM users WHERE first_name = $1', first_name) # select all users from the table with matching first name
        if not user:
            return None
        return dict(user)

    async def get_user_by_last_name(self, last_name):
        user = await self.fetchrow('SELECT * FROM users WHERE last_name = $1', last_name) # select all users from the table with matching last name
        if not user:
            return None
        return dict(user)

    async def get_admins(self):
        user = await self.fetch('SELECT * FROM users WHERE admin = true') # select all users from the table with admin status
        if not user:
            return None
        return user

    async def get_user_by_email(self, email):
        user = await self.fetchrow('SELECT * FROM users WHERE email = $1', email) # select all users from the table with matching email
        if not user:
            return None
        return dict(user)
```

```python

    async def edit_profile(self, id, user):
        await self.execute('UPDATE users SET first_name = $1, last_name = $2, email = $3, hashed_password = $4, admin = $5 WHERE id = $6',
                           user['first_name'], user['last_name'], user['email'],
                           hash_password(user['password']), user['admin'], id
                           ) # update users table by id with provided user dictionary
        return user

    async def create_user(self, user):
        await self.execute(
            'INSERT INTO users (first_name, last_name, email, hashed_password, admin) VALUES ($1, $2, $3, $4, $5)',
            user['first_name'], user['last_name'], user['email'],
            hash_password(user['password']), user['admin'] # create user in users table using provided user dictionary
        )
        return dict(user)

    async def delete_user(self, id):
        await self.execute(
            'DELETE FROM users WHERE id = $1', id
        ) # delete user from users table with provided id
        return id


    async def edit_user(self, id, user):
        await self.execute('UPDATE users SET first_name = $1, last_name = $2, email = $3, hashed_password = $4, admin = $5 WHERE id = $6',
                           user['first_name'], user['last_name'], user['email'],
                           hash_password(user['password']), user['admin'], id
                           ) # update users table by id with provided user dictionary
        return user

```

```
    async def create_table_users(self):
        await self.execute(
            'CREATE TABLE IF NOT EXISTS users (id serial PRIMARY KEY, first_name varchar(255), last_name varchar(255), email varchar(255), hashed_password varchar(255), admin boolean)'
        ) # create users table if it doesn't exist
```

api/db/schemas.py

The `schemas.py` file is used by `FastAPI` to validate data being passed to the program by users.

```
from pydantic import BaseModel # pydantic is a lightweight schema validation library for Python. it is used to validate the data that is passed to the API.

class UserBase(BaseModel): # defines the UserBase class. this class is used to validate the data that is passed to the API. it inherits from the BaseModel class
    email: str # defines the email field as a string
    admin: bool = False # defines the admin field as a boolean. this field is optional and defaults to false
    first_name: str = None # defines the first_name field as a string. this field is optional and defaults to None
    last_name: str = None # defines the last_name field as a string. this field is optional and defaults to None

class UserLogin(UserBase): # defines the UserLogin class
    password: str # defines the password field as a string

class UserCreate(UserBase): # defines the UserCreate class
    password: str # defines the password field as a string

class UserEdit(BaseModel): # defines the UserEdit class
    password: str # defines the password field as a string
    email: str = None # defines the email field as a string. this field is optional and defaults to None
    new_password: str = None # defines the new_password field as a string. this field is optional and defaults to None
    admin: bool = False # defines the admin field as a boolean. this field is optional and defaults to false
```

```
    first_name: str = None # defines the first_name field as a string. this field is optional and defaults to None
    last_name: str = None # defines the last_name field as a string. this field is optional and defaults to None

class User(UserBase):
    id: int # defines the id field as an integer

class TokenData(BaseModel):
    email: str = None # defines the email field as a string. this field is optional and defaults to None
    permissions: str = "user" # defines the permissions field as a string. this field is optional and defaults to "user"
```

api/utils/base64.py

This file contains a class for encoding and decoding base64.

```
class Encoding:
    def add_zeros(self, binary): # defines the add_zeros method
        if len(binary) % 8: # if the length of the binary string is not divisible by 8
            binary = '0' + binary # adds a zero to the beginning of the binary string
            return self.add_zeros(binary) # calls the method again recursively
        else: # otherwise
            return binary # returns the binary string

class Base64(Encoding): # creates a class called Base64 that inherits from Encoding

    def __init__(self): # initializes the class
        self.table = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/' # defines a table of allowed characters for base64 encoding
        self.__init__()

    def encode(self, text): # defines the encode method
        binary = '' # defines a variable to store the binary representation of the text

```

```
        for c in text: # iterates through the characters in the text
            binary += self.add_zeros(str(bin(ord(c)))[2:]) # converts the character to its binary representation and adds it to the binary string with the add_zeros method
        while len(binary) % 3: # while the length of the binary string is not divisible by 3
            binary += '00000000' # adds 8 zeros to the end of the binary string
        for i in range(6, len(binary) + int(len(binary) / 6), 7): # iterates through the binary string, starting at the 6th position, and every 7th position
            binary = binary[:i] + ' ' + binary[i:] # adds a space to the binary string
        binary = binary.split(' ') # splits the binary string into a list of strings at the space
        if '' in binary:  # if the list contains an empty string
            binary.remove('') # removes the empty string from the list
        base64 = '' # defines a variable to store the base64 representation of the text
        for b in binary: # iterates through the binary strings in the list
            if b == '000000': # if the binary string is equal to '000000'
                base64 += '=' # adds an equals sign to the base64 string
            else: # otherwise
                base64 += self.table[int(b, 2)] # converts the binary string to an integer and adds the character at the index of the integer to the base64 string
        return base64 # returns the base64 string

    def decode(self, text): # defines the decode method
        binary = '' # defines a variable to store the binary representation of the text
        for c in text: # iterates through the characters in the text
            if c == '=': # if the character is an equals sign
                binary += '000000' # adds 6 zeros to the binary string
            else: # otherwise
                binary += self.add_zeros(str(bin(self.table.
```

```
                    index(c)))[2:]) # converts the character to its binary representation and adds it to the binary string with the add_zeros method
        for i in range(8, len(binary) + int(len(binary) / 8), 9): # iterates through the binary string, starting at the 8th position, and every 9th position
            binary = binary[:i] + ' ' + binary[i:] # adds a space to the binary string
        binary = binary.split(' ') # splits the binary string into a list of strings at the space
        if '' in binary: # if the list contains an empty string
            binary.remove('') # removes the empty string from the list
        text = '' # defines a variable to store the text representation of the text
        for b in binary: # iterates through the binary strings in the list
            if not b == '00000000': # if the binary string is not equal to '00000000'
                text += chr(int(b, 2)) # converts the binary string to an integer and adds the character at the index of the integer to the text string
        return text # returns the text string
```

api/utils/sentiment.py

This class loads the sentiment analysis model trained on Google Collab, and makes it accessible through a class.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model

class Sentiment(Tokenizer): # class to predict sentiment of a text, inherits from keras.preprocessing.text.Tokenizer
    def __init__(self):
        self.model = load_model("model.hdf5") # load model
        super().__init__() # initialise class

    def predict(self, text): # predict sentiment of text
        sequence = self.texts_to_sequences([text]) # convert text to sequence of tokens
```

```
        t = pad_sequences(sequence, maxlen=200) # pad sequence to 200 tokens
        return self.model.predict(t) - 1 # return the models prediction for the text as a number between 1 and -1
```

server/rsa/keygen.py

This file is responsible for generating RSA keys. It produces two pairs of numbers, for public and private keys. I will need to use another program such as OpenSSL to convert to the two pairs into the commonly used .PEM format, so that my API server can use them. How the algorithm works is explained in the design section of this document.

```
import random


def miller_rabin(num):
    s = num - 1 # s is the number minus 1
    t = 0 # t is temporary variable set to 0

    while s % 2 == 0: # while s is even, divide by 2
        s = s // 2 # divide by 2
        t = t + 1 # increment t

    for x in range(40): # repeat 10 times, for 10 rounds
        a = random.randint(2, num - 1) # generate a random number between 2 and num - 1
        v = (a ** s) % num # calculate v = (a^s) mod num
        if v != 1: # if v is not 1,
            i = 0 # set i to 0
            while v != num - 1: # while v is not num - 1
                if i == t: # if i is equal to t,
                    return False # return false (not prime)
                else: # else
                    i = i + 1 # increment i
                    v = (v**2) % num # calculate v = (v^2) mod num
        return True # return true (prime)


def generate_prime(keysize):
    while True: # loop until we get a prime
        num = random.randint(2**(keysize-1),2**keysize) #
```

```
            generate a number of keysize bits
        if miller_rabin(num) == True: # if the number is prime, return it
            return num # keep on running loop until we generate a prime


def egcd(a, b):
    if a == 0: # in the case that a is 0, we need to return b, 0, 1
        return (b, 0, 1) # b is the gcd, 0 is x, 1 is y
    else:
        gcd, y, x = egcd(b % a, a) # recursively call the function, with the inputs b mod a, and a
        return (gcd, x - (b // a) * y, y) # return a tuple using some of the output of the egcd function. // is integer division.

# greatest common divisor
def gcd(a, b):
    return egcd(a, b)[0] # return the gcd of a and b

keysize = 8

p = generate_prime(keysize) # generate a prime of keysize 8
q = generate_prime(keysize) # generate a prime of keysize 8

n = p * q # n is the product of p and q

while True: # generate value of e until we get one that is coprime with n
    e = random.randint(2 ** (keysize - 1), 2 ** (keysize)) # generate a random number between 2^(keysize - 1) and 2^(keysize)
    if gcd(e, (p - 1) * (q - 1)) == 1: # if the gcd of e and (p - 1) * (q - 1) is 1,
        break # break the loop

g, x, y = egcd(e, (p - 1) * (q - 1)) # use the extended euclidean algorithm to find the gcd of e and (p - 1) * (q - 1)
d = x % ((p - 1) * (q - 1)) # d is the inverse of e mod (p - 1) * (q - 1)


```

```
publickey = (n, e) # public key is n and e
privatekey = (n, d) # private key is n and d

print("public key:", publickey) # print the public key
print("private key:", privatekey) # print the private key
```

server/news/update.py

This file will be run periodically. It fetches new news articles from the News API, and adds them to the database if they do not already exist. The News API imposes a rate limit, so it should not be run too frequently. A few times a day will suffice.

```
import requests
import os
from dotenv import load_dotenv
import asyncio
import asyncpg
import nest_asyncio

load_dotenv() # load the .env file
nest_asyncio.apply() # required to run the main function asynchronously

database_url = os.getenv("DATABASE_URL") # set database_url to the environment variable DATABASE_URL
api_key = os.getenv("NEWS_API_KEY") # set api_key to the environment variable NEWS_API_KEY

keywords = ['crypto','bitcoin','ethereum','dogecoin','cryptocurrency','nft','blockchain','defi'] # list of keywords to search the api for
api_url = 'https://newsapi.org/v2/everything?q=' + '%20OR%20'.join(keywords) + '&apiKey=' + api_key + '&pageSize=100&page=1' # api url made with the api key and the keywords

async def main(): # main function (asynchronous)
    conn = await asyncpg.connect(database_url) # connect to the database using the database_url
    req = requests.get(api_url) # request the api url and save the response to req variable

    count = await conn.fetchval('SELECT COUNT(*) FROM news') # store the number of rows in the news table in count
```

```
variable

    for item in req.json()['articles']: # iterate through each item in the articles array
        news = { # convert the item to a dictionary
                'publication': item['source']['name'],
                'author': item['author'],
                'title': item['title'],
                'description': item['description'],
                'url': item['url'],
                'imageUrl': item['urlToImage'],
                'date': item['publishedAt'],
                'content': item['content']
        }


        n = await conn.fetch('SELECT * FROM news WHERE title = $1', news['title']) # check if the news already exists in the database with matching title
        if not n: await conn.execute( # if the news does not exist in the database, insert it using the news dictionary
            'INSERT INTO news (publication, author, title, description, url, imageUrl, date, content) VALUES ($1, $2, $3, $4, $5, $6, $7, $8)',
            news['publication'],
            news['author'],
            news['title'],
            news['description'],
            news['url'],
            news['imageUrl'],
            news['date'],
            news['content']
        )




    new_count = await conn.fetchval('SELECT COUNT(*) FROM news') # count again the number of rows in the news table
    added = new_count - count # calculate the number of rows added
    print(added, "added.") # print the number of rows added
    await conn.close() # close the connection to the database

```

```
asyncio.get_event_loop().run_until_complete(main()) # run the main function in the event loop
```

server/sentiment/train.py

This file is responsible for creating the sentiment analysis model used by the application. This will be ran once on Google Collab, and will produce a model that can be downloaded for use by the API server.

```
import re
from gensim.utils import simple_preprocess
from sklearn.model_selection import train_test_split
import tensorflow as tf
import keras
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras import layers
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint
from nltk.tokenize.treebank import TreebankWordDetokenizer
# imports

def create_dataset(path): # function to load the data and creata dataframe with relevent columns
    dataset = pd.read_csv(path) # load the data into a pandas dataframe
    dataset = dataset.dropna() # drop rows with missing values
    dataset = dataset[['selected_text', 'sentiment']] # select the relevant columns
    return dataset # function to create the tokenizer

def create_labels(dataset): # function to create the labels
    labels = np.array(dataset['sentiment']) # create a numpy array of the labels from the sentiment column in the dataframe
    temp = [] # temporary variable
    for i in range(len(labels)): # loop through the labels
        if labels[i] == 'neutral': # if the label is neutral
            temp.append(0) # append a 0
        if labels[i] == 'negative': # if the label is negative
            temp.append(1) # append a 1
```

```
        if labels[i] == 'positive': # if the label is positive
            temp.append(2) # append a 2
    temp = np.array(temp) # convert the list to a numpy array
    labels = tf.keras.utils.to_categorical(temp, 3, dtype="float32") # convert the labels to categorical data with keras utils
    return labels # return the labels

def clean(data): # function to remove unnecessary characters from the text using Regex
    data = data.apply(lambda x: re.sub(r'http\S+', '', x)) # removes all urls
    data = data.apply(lambda x: re.sub(r'#\S+', '', x)) # remove hashtags
    data = data.apply(lambda x: re.sub(r'@\S+', '', x)) # removes @mentions
    data = data.apply(lambda x: re.sub(r'[^\w\s]', '', x)) # remove punctuation
    data = data.apply(lambda x: re.sub(r'\s+', ' ', x)) # remove multiple spaces
    data = data.apply(lambda x: re.sub(r"\'", "", x)) # removes single quotes
    return data

def data_to_words(dataset): # to return a list of lists of words
    sentences_temp = dataset['selected_text'] # get the selected text column
    temp = clean(sentences_temp) # clean the text using the clean function
    temp = temp.values.tolist() # convert the dataframe to a list
    for i in temp: # loop through the list
        yield(simple_preprocess(str(i), deacc=True)) # return a list of lists of words

def form_sentences(data_words): # function to form the sentences
    temp = [] # temporary variable
    for i in range(len(data_words)): # loop through the data
        temp.append(TreebankWordDetokenizer().detokenize(data_words[i])) # append the detokenized text to the list
    return np.array(temp) # return the list as a numpy array

def create_tokenizer(data): # function to create the tokenizer
    tokenizer = Tokenizer(num_words=5000) # create a tokenizer
```

```
        with 5000 words
    tokenizer.fit_on_texts(data) # fit the tokenizer on the data
    return tokenizer # return the tokenizer

def create_sequences(tokenizer, data): # function to create the sequences
    return tokenizer.texts_to_sequences(data) # return the sequences

def create_tweets(sequence): # function to create the tweets
    return pad_sequences(sequence, maxlen=200)

class LSTM(Sequential): # class to create the LSTM model
    def __init__(self, max_words, max_len, tokenizer): # initialise the model
        self.max_words = max_words # set the max words
        self.max_len = max_len # set the max length
        self.tokenizer = tokenizer # set the tokenizer
        super().__init__() # call the parent class
        self.add(layers.Embedding(self.max_words, 40, input_length=self.max_len)) # add an embedding layer
        self.add(layers.Bidirectional(layers.LSTM(20,dropout=0.6))) # add a bidirectional LSTM layer
        self.add(layers.Dense(3,activation='softmax')) # add a dense layer
        self.compile(optimizer='rmsprop',loss='categorical_crossentropy', metrics=['accuracy']) # compile the model

    def create_checkpoint(self): # function to create the checkpoint
        self.checkpoint = ModelCheckpoint("model.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto', period=1,save_weights_only=False) # create a checkpoint

    def train(self, trainx, trainy, epochs=70): # function to train the model
        return self.fit(trainx, trainy, epochs=epochs, validation_data=(testx, testy),callbacks=[self.checkpoint]) # train the model

    def load_model(self, path): # function to load the model
        keras.models.load_model(path) # load the model

```

```
    def evaluate_self(self, testx, testy): # function to evaluate the model
        return self.evaluate(testx, testy, verbose=2) # evaluate the model

    def predict_sentiment(self, text): # function to predict the sentiment
        sequence = self.tokenizer.texts_to_sequences([text]) # create the sequence
        temp = pad_sequences(sequence, maxlen=self.max_len) # pad the sequence
        return self.predict(temp) # predict the sentiment

dataset = create_dataset('dataset.csv') #
data_words = list(data_to_words(dataset) # create a list of lists of words
labels = create_labels(dataset) # create the labels
data = form_sentences(data_words) # form the sentences
tokenizer = create_tokenizer(data) # create the tokenizer
sequences = create_sequences(tokenizer, data) # create the sequences
tweets = create_tweets(sequences) # create the tweets
trainx, testx, trainy, testy = train_test_split(tweets, labels, random_state=1) # split the data into training and testing sets

model = LSTM(max_words=5000, max_len=200, tokenizer=tokenizer) # create the model
model.create_checkpoint() # create the checkpoint
model.train(trainx, trainy) # train the model
loss, accuracy = model.evaluate_self(testx, testy) # evaluate the model
```

client/pages/account/index.js

This file is the account homepage. Once users are logged in they will be directed here.

```
import useSWR from 'swr';
import Auth from '../../services/auth';
import WelcomeBanner from '../../components/account/welcome';
import { useRouter } from 'next/router';
import Loading from '../../components/loading';
import Comments from '../../components/comments';
```

```
7
8  function Dashboard() {
9    const router = useRouter(); // this creates an instance of the
         router, which allows manipulation of the url
10   const auth = new Auth(); // this creates an instance of the
        auth service
11   const { data, loading, error } = useSWR(['/users/me', true],
        auth.fetcher); // this fetches the user data from the api,
        with the fetcher function
12   if (error || loading) { // if there is an error or the data is
         loading, redirect to the login page and display the
        loading component
13     router.push('/login'); // redirect to the login page
14     return <Loading />; // display the loading component
15   }
16   if (data && data.first_name) { // if there is data and it
        contains a first name, display the welcome banner
17     return (
18       <>
19         <WelcomeBanner line1={'Welcome back, ' + data.first_name
               + ' ' + data.last_name} /> // display the welcome
             banner
20
21         <p className="">Email: {data.email}</p> // display the
             user's email
22         <button // button to logout the user
23           onClick={() => { // on click, logout the user
24             auth.deleteToken(); // delete the authentication
                 token from local storage
25             router.reload(); // reload the page
26           }}
27           type="button"
28           className="mt-5 bg-complementary-800 text-white
                rounded px-2 py-1 transition duration-200 ease
                select-none hover:bg-complementary-900 focus:
                outline-none focus:shadow-outline"
29         >
30           Logout
31         </button>
32         <div className="mt-8">
33           {data.comments && ( // if the user has comments,
                display the comments
34             <>
35               <h3 className="text-2xl font-semibold">Your
                    Comments</h3>
```

```
                <Comments comments={data.comments} /> // display
                    the comments
              </>
            )}
          </div>
        </>
      );
    } else {
      return <Loading />;
    }
  }

  export default Dashboard;
```

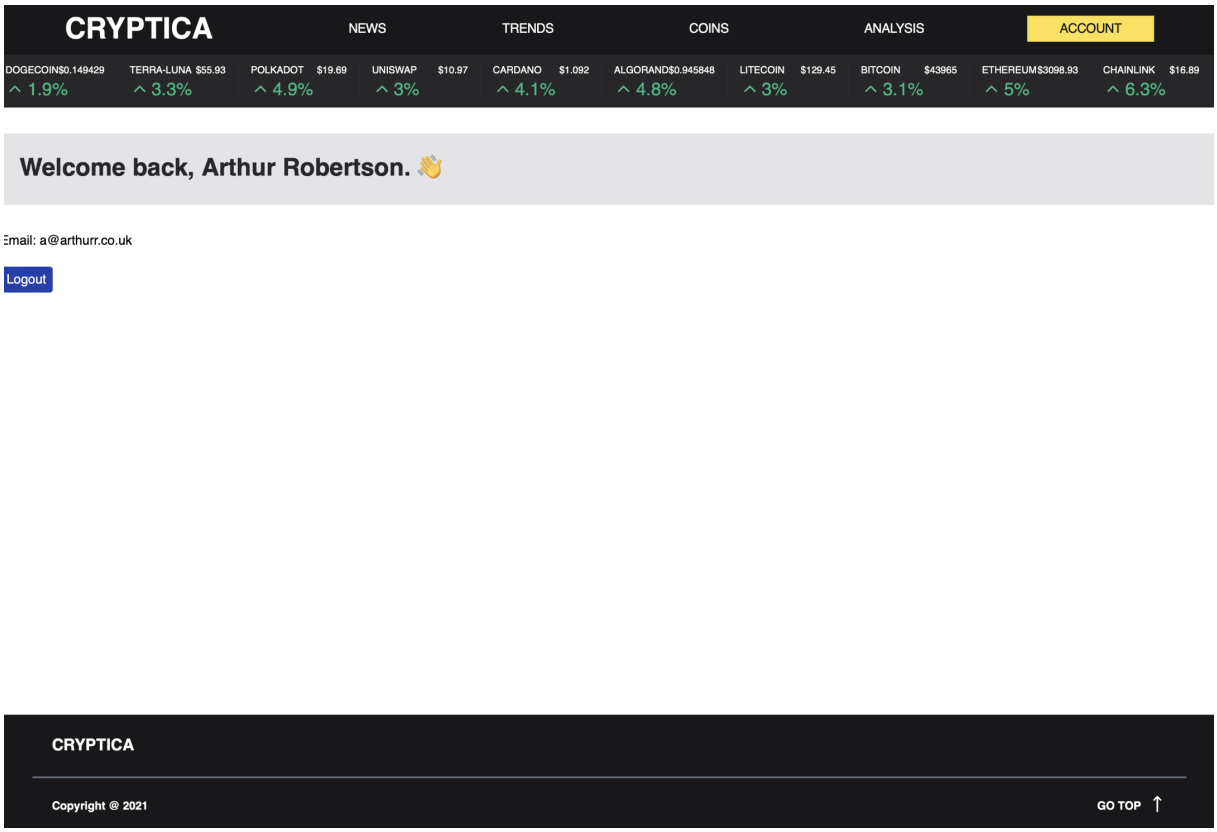

Figure 25: Screenshot of the accounts page

client/pages/tweet-analysis/index.js

This page is used for analysing specific tweets from a user. By default, it has an input field for the user to enter a twitter handle, and a specified cryptocoin. Then after submitting a handle, it displays a list of that users tweets that mentions the selected cryptocurrency. The user is then free to click on any of the tweets on the side which will then bring up a graph showing the price of the cryptocurrency during the tweet.

```
import Auth from '../../services/auth';
import { useEffect, useState } from 'react';
import Loading from '../../components/loading';
import OHCL from '../../components/analysis/ohcl';
import useUser from '../../services/user';
import Tweet from '../../components/analysis/tweet';

export default function Analysis() {
  const auth = new Auth(); // this creates an instance of the auth service
  const [data, setData] = useState(); // this creates a state variable for the data
  const [price, setPrice] = useState(); // this creates a state variable for the price
  const [index, setIndex] = useState(0); // this creates a state variable for the index
  const [prevIndex, setPrevIndex] = useState(); // this creates a state variable for the previous index with default value 0
  const [loading, setLoading] = useState(false); // this creates a state variable for the loading state
  const [coin, setCoin] = useState(); // this creates a state variable for the coin

  function convertDate(date) { // this function converts the date to UNIX timestamp
    const dateArray = date.split(' '); // split the date into an array
    const dateString = dateArray[0];
    const timeString = dateArray[1];
    const dateArray2 = dateString.split('-'); //
    const timeArray = timeString.split(':');
    const year = parseInt(dateArray2[0]);
    const month = parseInt(dateArray2[1]);
    const day = parseInt(dateArray2[2]);
```

```
26      const hour = parseInt(timeArray[0]);
27      const minute = parseInt(timeArray[1]);
28      const second = parseInt(timeArray[2]);
29      const d = new Date(year, month - 1, day, hour, minute,
           second); // create a new date object with the date and
           time from the previous steps
30      return d.getTime() / 1000; // return the UNIX timestamp
31    }
32
33    const { user } = useUser(); // this uses the userUser hook to
         get the user data
34
35    const submitForm = (form) => { // this function submits the
         form
36      form.preventDefault(); // this disables the default form
           submission behavior, which is to refresh the page upon
           form submission
37      setLoading(true); // set the loading state to true
38      setCoin(form.target.coin.value); // set the coin state to
           the value of the coin input
39      setIndex(); // set the index state to 0
40      auth // this fetches the data from the api
41        .fetcher(  // this fetches the data from the api
42          `/twitter/search?username=${form.target.twitterhandle.
               value}&coin=${form.target.coin.value}`, // this is
               the url to fetch the data from
43          true
44        )
45        .then((res) => { // this is the response from the api
46          setData(res); // set the data state to the response
47        })
48        .then(() => { // this is the response from the api
49          setLoading(false); // set the loading state to false
50        });
51    };
52
53    const convert = (coin) => { // this function converts the coin
         to ticker format
54      switch (coin) { // switch on the coin
55        case 'Bitcoin':
56          return 'BTCUSDT';
57        case 'Ethereum':
58          return 'ETHUSDT';
59        case 'Doge':
60          return 'DOGEUSDT';
```

```
      case 'Litecoin':
        return 'LTCUSDT';
      case 'Cardano':
        return 'ADAUSDT';
    }
  };

  useEffect(() => { // this is the effect to run when the component mounts
    if (!user) { // if the user is not logged in
      return; // return nothing
    }

    if (data && data[0] && index !== prevIndex && data[0][index] && data[0][index].datetime) { // if the data is not empty and the index is not the same as the previous index and the data has a datetime
      setLoading(true); // set the loading state to true
      auth
        .fetcher(`/crypto/${convert(coin)}/${convertDate(data[0][index].datetime)}`) // this fetches the price from the api
        .then((res) => { // this is the response from the api
          setPrice(res); // set the price state to the response
        })
        .then(() => {
          setLoading(false); // set the loading state to false
        });
      setPrevIndex(index); // set the previous index state to the index
    }
  }, [user, data, index]); // this is the effect to run when the component mounts

  if (!user) { // if the user is not logged in
    return 'You need to login to access this page!'; // return the message
  }

  return ( // this is the default return statement
    <>
      <div className="flex px-5">
        <div className="h-1/2 top-0 sticky w-2/3 flex mr-6">
          <div className="flex-1">
            <div className="py-5 text-6xl text-center font-
```

```
                    semibold lg:text-left transform">
                  ANALYSIS
                </div>
                <form onSubmit={submitForm} className="bg-complementary-100 p-5 mb-6 space-x-5 flex">
                  <div className="relative inline-flex self-center flex-initial">
                    <svg
                      className="text-white bg-primary-700 absolute top-0 right-0 m-2 pointer-events-none p-2 rounded"
                      xmlns="http://www.w3.org/2000/svg"
                      width="40px"
                      height="40px"
                      viewBox="0 0 38 22"
                      version="1.1"
                    >
                      <g stroke="none" strokeWidth="1" fill="none" fillRule="evenodd">
                        <g
                          transform="translate(-539.000000, -199.000000)"
                          fill="#ffffff"
                          fillRule="nonzero"
                        >
                          <g
                            id="Icon-/-ArrowRight-Copy-2"
                            transform="translate(538.000000, 183.521208)"
                          >
                            <polygon
                              id="Path-Copy"
                              transform="translate(20.000000, 18.384776) rotate(135.000000) translate(-20.000000, -18.384776) "
                              points="33 5.38477631 33 31.3847763 29 31.3847763 28.999 9.38379168 7 9.38477631 7 5.38477631"
                            />
                          </g>
                        </g>
                      </g>
                    </svg>
                    <select
                      id="coin"
```

```
                    className="text-xl font-bold rounded border-2 border-primary-700 text-neutral-600 h-14 w-44 pl-5 pr-10 bg-white focus:outline-none appearance-none"
                  >
                    <option>Bitcoin</option>
                    <option>Ethereum</option>
                    <option>Doge</option>
                    <option>Litecoin</option>
                    <option>Cardano</option>
                  </select>
                </div>
                <input
                  id="twitterhandle"
                  type="text"
                  required
                  placeholder="elonmusk"
                  className="flex-auto text-xl font-bold rounded border-2 border-primary-700 text-neutral-600 h-14 pl-5 pr-10 bg-white focus:border-neutral-400 focus:outline-none appearance-none"
                />
                <button
                  type="submit"
                  className="text-xl font-bold rounded text-white h-14 px-8 bg-primary-800 hover:bg-primary-900 focus:outline-none appearance-none"
                >
                  Submit
                </button>
              </form>

              {!loading && price && ( // if the loading state is false and the price state is not empty
                <>
                  <div className="py-3 text-4xl text-center lg:text-left font-medium transform">
                    {coin} at {data[0][index].datetime}
                  </div>
                  <div className="bg-neutral-100 h-96">
                    {price && price[29] && <OHCL data={price} />}
                  </div>
                  <div className="bg-neutral-100 p-5 my-8 h-full">
                    <div className="flex flex-col">
                      <div className="flex-1">
```

```
                        <div className="text-2xl text-center lg:text-left font-medium transform">
                          {coin}
                        </div>
                        <div className="text-xl text-center lg:text-left font-medium transform">
                          {data[0][index].datetime}
                        </div>
                        <div className="text-xl text-center lg:text-left font-medium transform">
                          {data[0][index].tweet}
                        </div>
                        <div className="text-xl text-center lg:text-left font-medium transform">
                          Sentiment: {data[0][index].sentiment}
                        </div>
                        <div className="text-xl text-center lg:text-left font-medium transform">
                          {(-(price[29][1] - price[34][1]) / price[29][1]) * 100}%
                        </div>
                      </div>
                    </div>
                  </div>
                </>
              )}
              {loading && <Loading />}
            </div>
          </div>

          <div className="flex-grow w-1/3">
            <h2 className="pb-2 pt-8 text-4xl leading-tight md:text-4xl">Tweets</h2>

            {data && data[0] && !data[0][0] && 'No results found' /* if the data state is not empty and the first index is empty, return the message */}
            {data &&
              data[0] &&
              data[0].map((item, key) => { // this is the map function to map the data state to the tweets
                return (
                  <button
                    key={key}
                    className="hover:bg-complementary-100
```

```
                        appearance-none w-full text-left border
                        border-neutral-300 dark:border-neutral-800
                        px-6 py-4 my-4 transition duration-500 ease
                        -in-out transform hover:-translate-y-1
                        hover:scale-105"
                    onClick={() => setIndex(key)}
                  >
                    <Tweet user={data[1]} tweet={item} />
                  </button>
                );
              })}
          </div>
        </div>
      </>
    );
  }
```

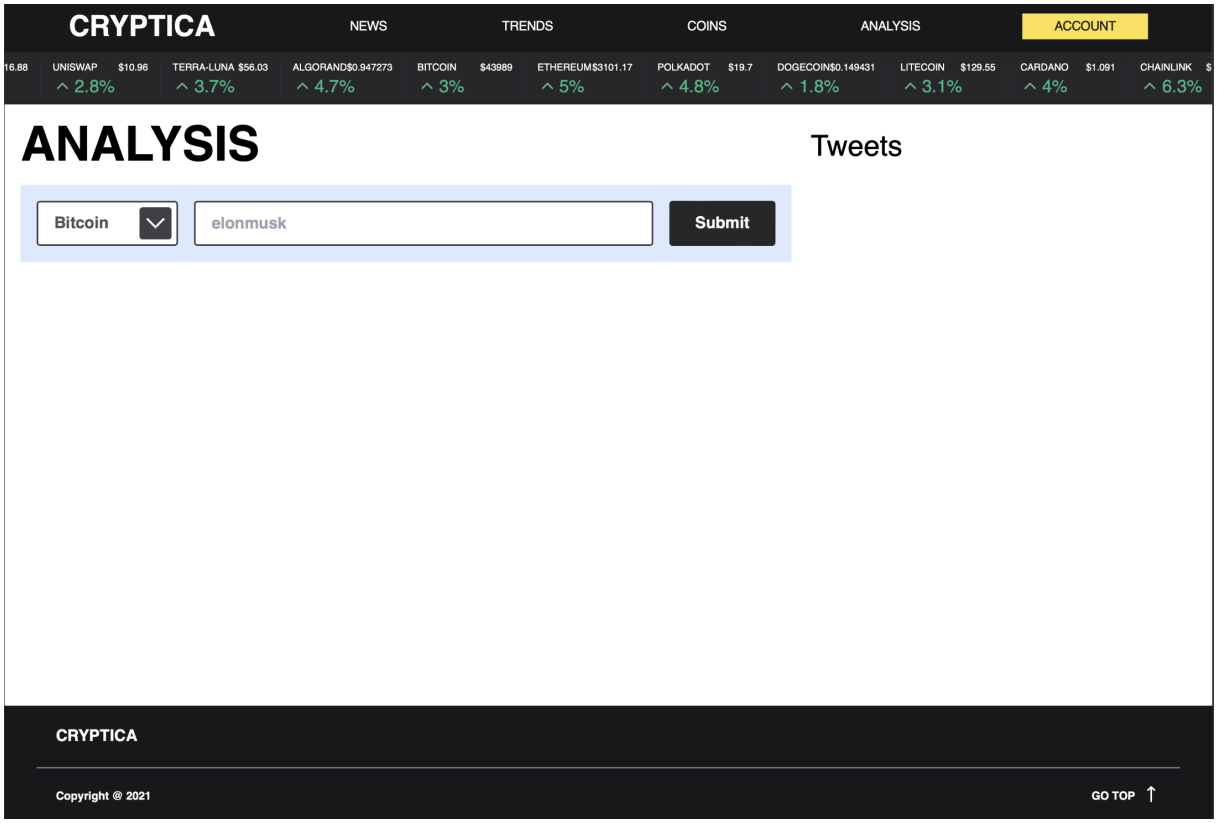

Figure 26: The page when the user first enters it, with the input fields.

Figure 27: Select Option allowing the choice of Cryptocurrency

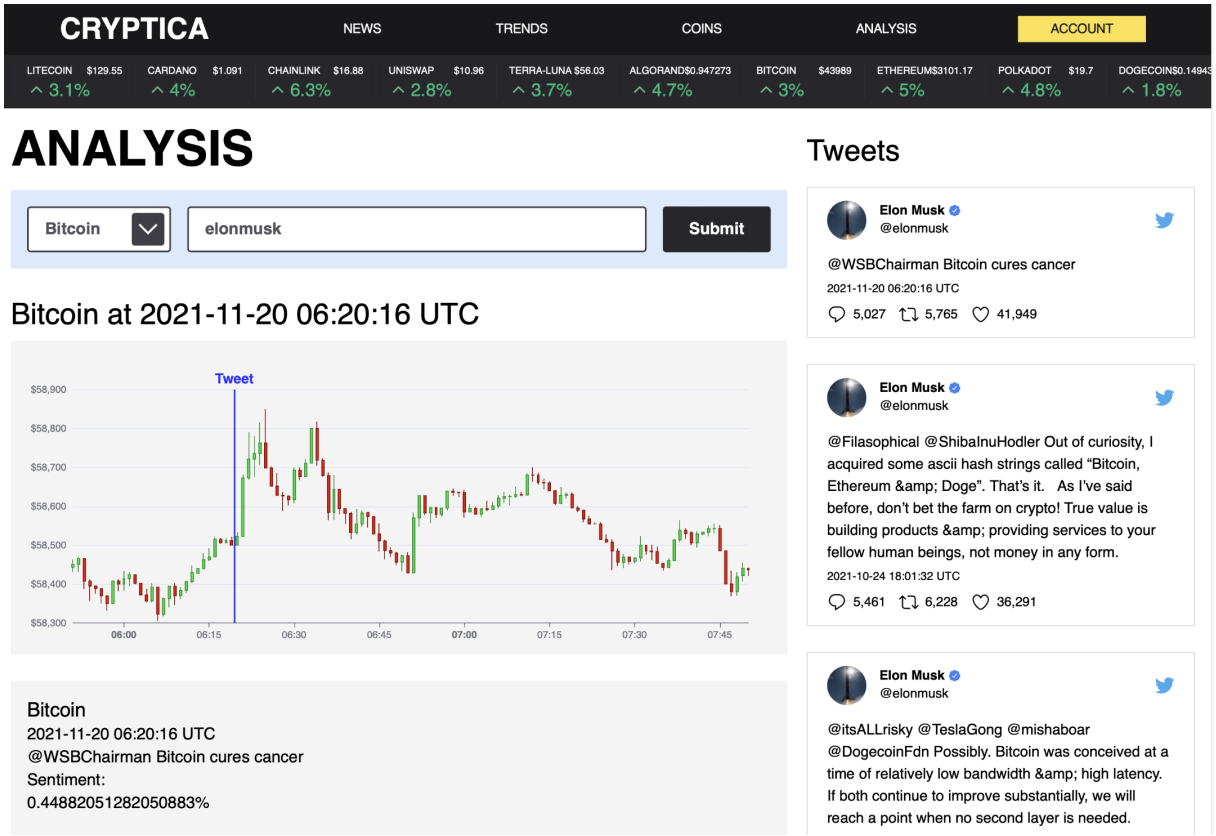

Figure 28: Full page with tweet selected from list of side.

client/pages/coin/index.js

This page contains a leaderboard of the top 50 coins ordered by marketcap, with data fetched from an external API.

```
import TableItem from '../../components/coin/tableitem';
import { useState, useEffect } from 'react';
import axios from 'axios';
import Loading from '../../components/loading';
const Coins = () => {
  const [data, setData] = useState(); // this creates a state variable for the data
  const [loading, setLoading] = useState(true); // this creates a state variable for the loading state

  const formatNumber = (num) => { // this function formats the number to a currency format
```

```
    if (num >= 1000000000000) { // 1 trillion to T
      return (num / 1000000000000).toFixed(1) + 'T';
    } else if (num >= 1000000000) { // 1 billion to B
      return (num / 1000000000).toFixed(1) + 'B';
    } else if (num >= 1000000) { // 1 million to M
      return (num / 1000000).toFixed(1) + 'M';
    } else if (num >= 1000) { // 1 thousand to K
      return (num / 1000).toFixed(1) + 'K';
    } else { // less than 1 thousand
      return num; // return the number
    }
  };

  useEffect(() => { // this is the effect to fetch the data
    setLoading(true); // set the loading state to true
    axios
      .get( // this fetches the data from the api
        'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=50&page=1&sparkline=false&price_change_percentage=24h'
      ) // this is the url to fetch the data from
      .then((res) => setData(res.data)) // this is the response from the api
      .then(() => setLoading(false))
      .catch((err) => err);
  }, []);
  if (loading || !data) return <Loading />; // if the loading state is true or the data state is undefined, return the loading component
  return (
    <div className="px-10 ">
      <div className="py-5 space-y-2">
        <h1 className="text-5xl">Coin Leaderboard</h1>
        <p>
          This page contains a list of the top 50 coins, ordered by marketcap. Click a coin to view
          more info.
        </p>
      </div>

      <div class="">
        <table class="w-full text-left">
  <thead>
    <tr class="text-neutral-400">
      <th class="font-normal px-3 pt-0 pb-3 border-b border-
```

```
            neutral-200 dark:border-neutral-800">
49 #
50        </th>
51        <th className="font-normal px-3 pt-0 pb-3 border-b border-
            neutral-200 dark:border-neutral-800">
52 Coin
53        </th>
54        <th className="font-normal px-3 pt-0 pb-3 border-b border-
            neutral-200 dark:border-neutral-800">
55 Price
56        </th>
57        <th className="font-normal px-3 pt-0 pb-3 border-b border-
            neutral-200 dark:border-neutral-800">
58 24h Change
59        </th>
60        <th className="font-normal px-3 pt-0 pb-3 border-b border-
            neutral-200 dark:border-neutral-800">
61 24h High
62        </th>
63        <th className="font-normal px-3 pt-0 pb-3 border-b border-
            neutral-200 dark:border-neutral-800">
64 24h Low
65        </th>
66        <th className="font-normal px-3 pt-0 pb-3 border-b border-
            neutral-200 dark:border-neutral-800">
67 Market Cap
68        </th>
69      </tr>
70    </thead>
71    {data.map((item) => ( // this is the map function to loop
        through the data and create the table with each of the
        items in the data
72      <TableItem // this is the table item component. data is
          passed in as props from the map function
73        key={item.market_cap_rank}
74        id={item.id}
75        image={item.image}
76        rank={item.market_cap_rank}
77        name={item.name}
78        symbol={item.symbol.toUpperCase()}
79        price={item.current_price}
80        change={item.price_change_percentage_24h}
81        high={item.high_24h}
82        low={item.low_24h}
83        marketcap={formatNumber(item.market_cap)}
```

```
      />
    ))}
</table>
        </div>
      </div>
    );
};

export default Coins;
```

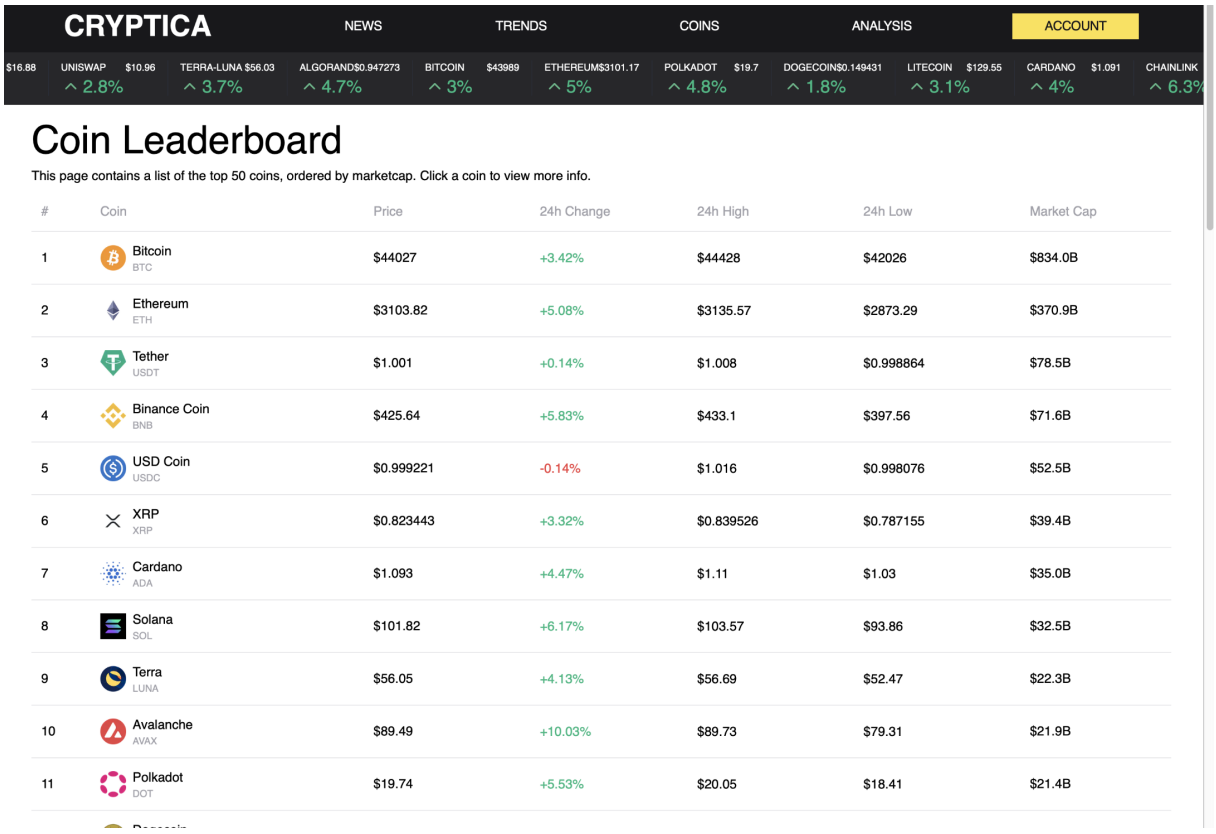

Figure 29: Coin Leaderboard Page Screenshot

client/page/coin/[id].js

This page is for displaying advanced details about a specific coin. It is accessible from the coins leaderboard page, and fetches data from an external api.

```
import { useRouter } from 'next/router';
import RelatedNews from '../../components/coin/relatednews';
```

```
3  import Graph from '../../components/coin/graph';
4  import axios from 'axios';
5  import { useEffect, useState } from 'react';
6  import Loading from '../../components/loading';
7  import Auth from '../../services/auth';
8  const Coin = () => {
9    const router = useRouter(); // this creates an instance of the
       router service
10   const [data, setData] = useState(); // this creates a state
       variable for the data
11   const [loading, setLoading] = useState(true); // this creates
       a state variable for the loading state
12   const [fav, setFav] = useState(false); // this creates a state
       variable for the favorite state
13   const [time, setTime] = useState('7d'); // this creates a
       state variable for the time
14   const [news, setNews] = useState(); // this creates a state
       variable for the news
15   const auth = new Auth(); // this creates an instance of the
       auth service
16
17   const formatNumber = (num) => { // this function formats the
       number
18     if (num >= 10 ** 12) {
19       return (num / 10 ** 12).toFixed(1) + 'T';
20     } else if (num >= 10 ** 9) {
21       return (num / 10 ** 9).toFixed(1) + 'B';
22     } else if (num >= 10 ** 6) {
23       return (num / 10 ** 6).toFixed(1) + 'M';
24     } else if (num >= 10 ** 3) {
25       return (num / 10 ** 3).toFixed(1) + 'K';
26     } else {
27       return num;
28     }
29   };
30
31   const buttonClick = () => { // this is the function to change
       the fav variable state to the opposite value
32     setFav(!fav);
33   };
34
35   useEffect(() => { // this is the effect to fetch the data
36     async function fetchNews() { // this is the function to
         fetch the news
37       return await auth.poster('/news/search', { // this is the
```

```
            url to fetch the data from
        phrase: router.query.coin // this is the post body to send to the api. router.query.coin is the coin name which is passed in from the url
      });
    }

    setLoading(true); // set the loading state to true
    if (!router.query.coin) { // if the coin is undefined, redirect to the home page
      return; // return the function
    }
    axios // this is the axios instance to fetch the data
      .get( // this fetches the data from the api
        `https://api.coingecko.com/api/v3/coins/${router.query.coin}?localization=false&tickers=false&community_data=false&developer_data=false&sparkline=false`
      ) // this is the url to fetch the data from
      .then((res) => setData(res.data)) // this sets the data state variable to the data
      .then(() => setLoading(false)) // this sets the loading state to false
      .catch((err) => err); // this catches any errors
    fetchNews().then((res) => setNews(res.data)); // this fetches the news
  }, [router.query.coin]); // this is the dependency array, which is the coin query

  if (router.isFallback || !router.query.coin || !data || loading) { // return loading in the following conditions
    return <Loading />;
  }
  return ( // return the following
    <>
      <div className="flex px-10">
        <div className={news ? 'pt-10 w-full lg:w-3/4' : 'pt-10 w-full'}>
          <div className="space-x-4">
            <span className="text-5xl font-medium">{data.name}</span>
            <span className="text-4xl font-medium text-neutral-400">
              {data.symbol.toUpperCase()}
            </span>
          </div>
```

```
            <div className="flex bg-neutral-50 mt-5 font-light">
              <div className="flex-auto p-5">
                <h3 className="text-sm">PRICE</h3>
                <h2 className="text-4xl font-medium">$ {data.market_data.current_price.usd}</h2>
              </div>
              <div className="flex-auto p-5">
                <h3 className="text-sm">24HR PRICE CHANGE</h3>
                <h2 className="text-2xl text-green-500 font-medium">
                  {Math.round(100 * data.market_data.price_change_percentage_24h) / 100}% {/* this is the percent change */}
                </h2>
              </div>
              <div className="flex-auto p-5">
                <h3 className="text-sm">MARKET CAP</h3>
                <h2 className="text-2xl font-medium">
                  ${formatNumber(data.market_data.market_cap.usd)} {/* this is the market cap from the API*/}
                </h2>
              </div>
              <div className="flex-auto p-5">
                <h3 className="text-sm">24HR MARKET CAP CHANGE</h3>
                <h2 className="text-2xl font-medium text-green-500">
                  {Math.round(
                    100 * data.market_data.market_cap_change_percentage_24h_in_currency.usd
                  ) / 100} {/* this is the market cap change from the API */}
                  %
                </h2>
              </div>

              <button
                onClick={buttonClick}
                className="flex-none p-2 text-center text-md bg-red-100 order-last hover:bg-red-200 w-16"
              >
                <svg viewBox="0 0 512 512">
                  {fav && (
                    <path
```

```
                    fill="red"
                    d="M376,30c
                       -27.783,0-53.255,8.804-75.707,26.168c
                       -21.525,16.647-35.856,37.85-44.293,53.268
            c-8.437-15.419-22.768-36.621-44.293-53.268C189
               .255,38.804,163.783,30,136,30C58
               .468,30,0,93.417,0,177.514
            c0,90.854,72.943,153.015,183.369,247.118c18
               .752,15.981,40.007,34.095,62.099,53.414C248
               .38,480.596,252.12,482,256,482
            s7.62-1.404,10.532-3.953c22
               .094-19.322,43.348-37.435,62.111-53.425C439
               .057,330.529,512,268.368,512,177.514
            C512,93.417,453.532,30,376,30z"
                  />
                )}
                <path
                  d="M474.644,74.27C449
                     .391,45.616,414.358,29.836,376,29.836c
                     -53.948,0-88.103,32.22-107.255,59.25
            c-4.969,7.014-9.196,14.047-12.745,20.665c
               -3.549-6.618-7.775-13.651-12.745-20.665c
               -19.152-27.03-53.307-59.25-107.255-59.25
            c-38.358,0-73.391,15.781-98.645,44.435C13
               .267,101.605,0,138.213,0,177.351c0
               ,42.603,16.633,82.228,52.345,124.7
            c31.917,37.96,77.834,77.088,131.005,122.397c19
               .813,16.884,40.302,34.344,62.115,53.429l0
               .655,0.574
            c2.828,2.476,6.354,3.713,9.88,3.713s7
               .052-1.238,9.88-3.713l0.655-0.574c21
               .813-19.085,42.302-36.544,62.118-53.431
            c53.168-45.306,99.085-84.434,131.002-122.395C495
               .367,259.578,512,219.954,512,177.351
            C512,138.213,498.733,101.605,474.644,74.27z M309
               .193,401.614c
               -17.08,14.554-34.658,29.533-53.193,45.646
            c-18.534-16.111-36.113-31.091-53.196-45.648C98
               .745,312.939,30,254.358,30,177.351c0
               -31.83,10.605-61.394,29.862-83.245
            C79.34,72.007,106.379,59.836,136,59.836c41
               .129,0,67.716,25.338,82.776,46.594c13
               .509,19.064,20.558,38.282,22.962,45.659
            c2.011,6.175,7.768,10.354,14.262,10.354c6
               .494,0,12.251-4.179,14.262-10.354c2
```

```
                .404-7.377,9.453-26.595,22.962-45.66
            c15.06-21.255,41.647-46.593,82.776-46.593c29
                .621,0,56.66,12.171,76.137,34.27C471
                .395,115.957,482,145.521,482,177.351
            C482,254.358,413.255,312.939,309.193,401.614z"
                />
              </svg>
            </button>
          </div>
          <div className="w-full bg-neutral-50 mt-8 h-96">
            <Graph coin={router.query.coin} time={time} /> {/*
                this is the graph component */}
          </div>
          {data.description.en && (
            <div
              className="w-full bg-neutral-50 mt-8 p-4"
              dangerouslySetInnerHTML={{ __html: data.
                  description.en }} {/* this is the description
                  from the API */}
            />
          )}
        </div>
        {news && (
          <div className="pl-10 h-1/2 top-0 sticky w-1/4 hidden
              lg:flex">
            <RelatedNews news={news} />
          </div>
        )}
      </div>
    </>
  );
};

export default Coin;
```

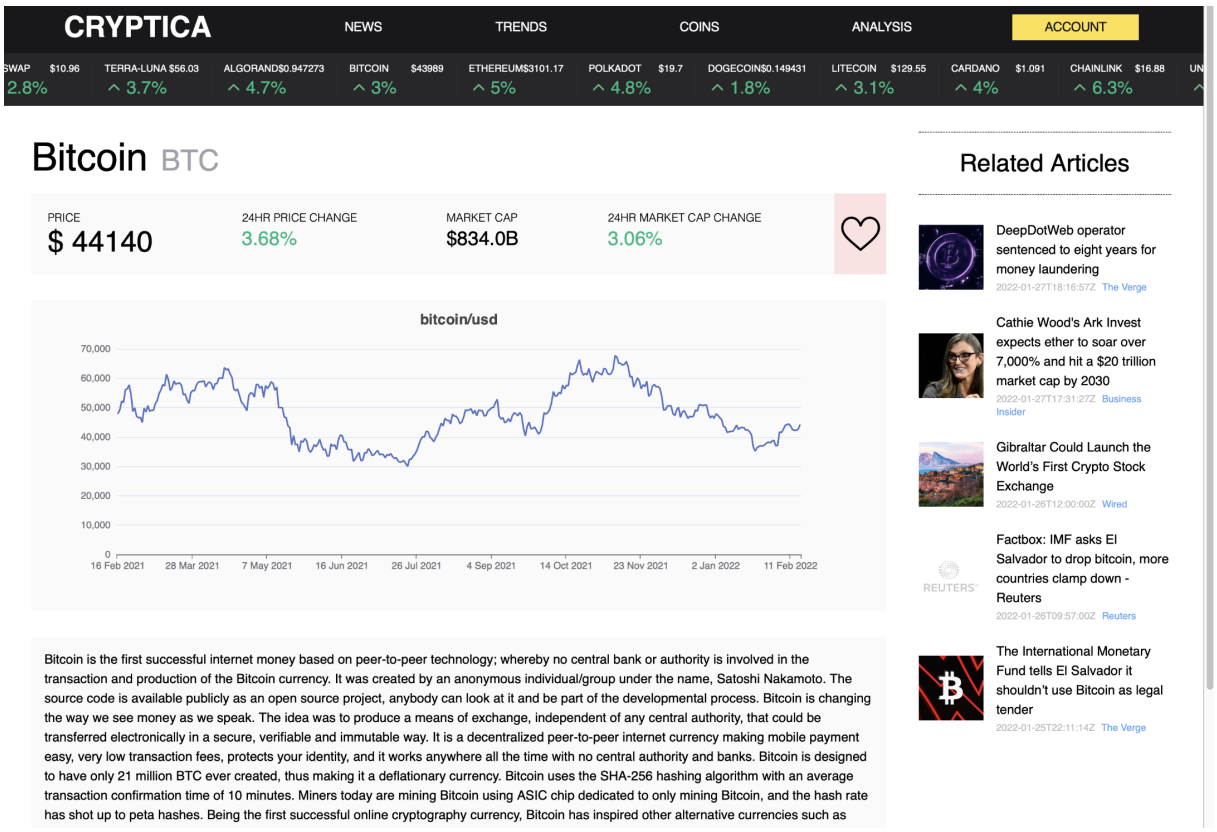

Figure 30: Bitcoin's page

client/page/account-analysis/index.js

This file is for analysising users twitters account on a general level. It has a field for inputting a user's twitter handle and for specifying a quantity of tweets to analyse. It produces several graphs and charts.

```
import useUser from '../../services/user';
import Auth from '../../services/auth';
import { useState } from 'react';
import Heatmap from '../../components/admin/heatmap';
import Piechart from '../../components/admin/piechart';
import Profile from '../../components/admin/profile';

const Admin = () => {
  const { user, loading } = useUser(); // this uses the userUser hook to get the user data
  const auth = new Auth(); // this creates an instance of the
```

```
    auth service
  const [data, setData] = useState(); // this creates a state variable for the data
  const [load, setLoading] = useState(false); // this creates a state variable for the loading state

  if (!user && !loading) { // if the user is not logged in and the loading state is false
    return <div>Unauthorised</div>;
  }

  const submitForm = (form) => { // this function submits the form
    form.preventDefault(); // this disables the default form submission behavior, which is to refresh the page upon form submission
    setLoading(true); // set the loading state to true
    auth // this fetches the data from the api
      .fetcher(
        `/twitter/search?username=${form.target.handle.value}&limit=${form.target.count.value}`,
        true // this is the url to fetch the data from
      )
      .then((res) => {
        setData(res);
      })
      .then(() => {
        setLoading(false);
      });
  };

  return (
    <div className={'bg-neutral-50 ' + (load ? 'cursor-wait' : ' ')}>
      <div className="grid justify-items-center py-5">
        <form onSubmit={submitForm} className="bg-complementary-100 p-5 space-x-5 flex w-3/4 ">
          <div className="relative inline-flex self-center flex-initial">
            <svg
              className="text-white bg-primary-700 absolute top-0 right-0 m-2 pointer-events-none p-2 rounded"
              xmlns="http://www.w3.org/2000/svg"
              width="40px"
              height="40px"
```

```
44                viewBox="0 0 38 22"
45                version="1.1"
46              >
47                <g stroke="none" strokeWidth="1" fill="none"
                     fillRule="evenodd">
48                  <g
49                    transform="translate(-539.000000, -199.000000)
                        "
50                    fill="#ffffff"
51                    fillRule="nonzero"
52                  >
53                    <g id="Icon-/-ArrowRight-Copy-2" transform="
                        translate(538.000000, 183.521208)">
54                      <polygon
55                        id="Path-Copy"
56                        transform="translate(20.000000, 18.384776)
                            rotate(135.000000) translate
                           (-20.000000, -18.384776) "
57                        points="33 5.38477631 33 31.3847763 29
                           31.3847763 28.999 9.38379168 7
                           9.38477631 7 5.38477631"
58                      />
59                    </g>
60                  </g>
61                </g>
62              </svg>
63              <select
64                id="count"
65                className="text-xl font-bold rounded border-2
                     border-primary-700 text-neutral-600 h-14 w-52
                     pl-5 pr-10 bg-white focus:outline-none
                     appearance-none"
66              >
67                <option value="100">100 Tweets</option>
68                <option value="250">250 Tweets</option>
69                <option value="500">500 Tweets</option>
70                <option value="1000">1000 Tweets</option>
71                <option value="2000">2000 Tweets</option>
72              </select>
73            </div>
74            <input
75              id="handle"
76              type="text"
77              required
78              placeholder="elonmusk"
```

```
            className="flex-auto text-xl font-bold rounded border-2 border-primary-700 text-neutral-600 h-14 pl-5 pr-10 bg-white focus:border-neutral-400 focus:outline-none appearance-none"
          />
          <button
            type="submit"
            disabled={load}
            className={
              'text-xl font-bold rounded text-white h-14 px-8 bg-primary-800 hover:bg-primary-900 focus:outline-none appearance-none' +
              (load ? ' opacity-50 cursor-not-allowed' : '')
            }
          >
            Submit
          </button>
        </form>
      </div>
      {data && data[0] && (
        <>
          <div className="flex flex-wrap overfull-hidden">
            <div className="bg-green-50 w-full lg:w-2/3">
              <Profile data={data} />
            </div>
            <div className="w-full lg:w-1/3">
              <div className="bg-red-50 pt-5 pb-2 px-2">
                <h1 className="text-center text-2xl font-bold text-neutral-800">Devices Used</h1>
              </div>
              <div className="bg-red-50 h-72  px-2">
                <Piechart
                  data={data[0].map((tweet) => {
                    return tweet.source; // this maps the data to the source property
                  })}
                />
              </div>
            </div>
          </div>
          <div className="bg-complementary-50 py-5">
            <h1 className="text-center text-2xl font-bold text-neutral-800">
              Tweets at times across the week (GMT)
            </h1>
```

```
            </div>
            <div className="bg-complementary-50 h-96">
              <Heatmap
                data={data[0].map((tweet) => {
                  return tweet.datetime; // this maps the data to
                     the datetime property
                })}
              />
            </div>
          </>
        )}
      </div>
    );
};

export default Admin;
```

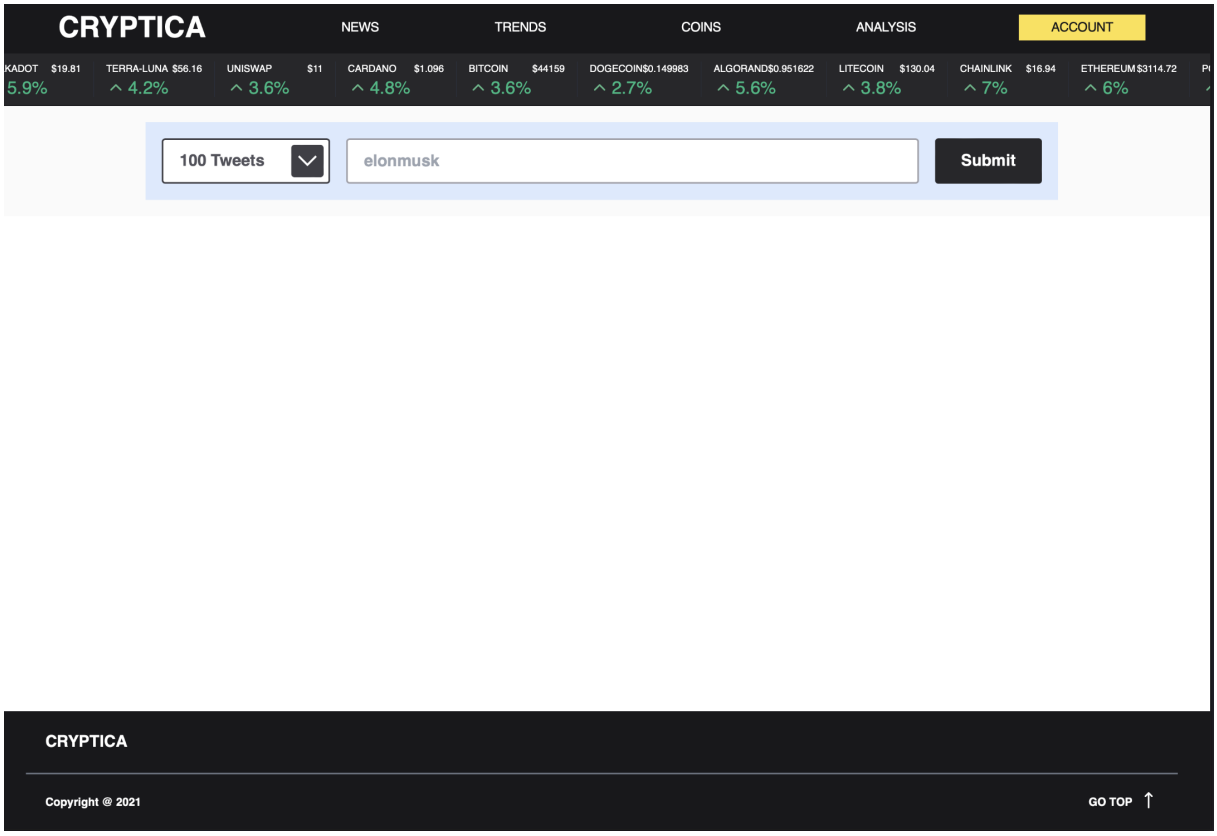

Figure 31: The input field

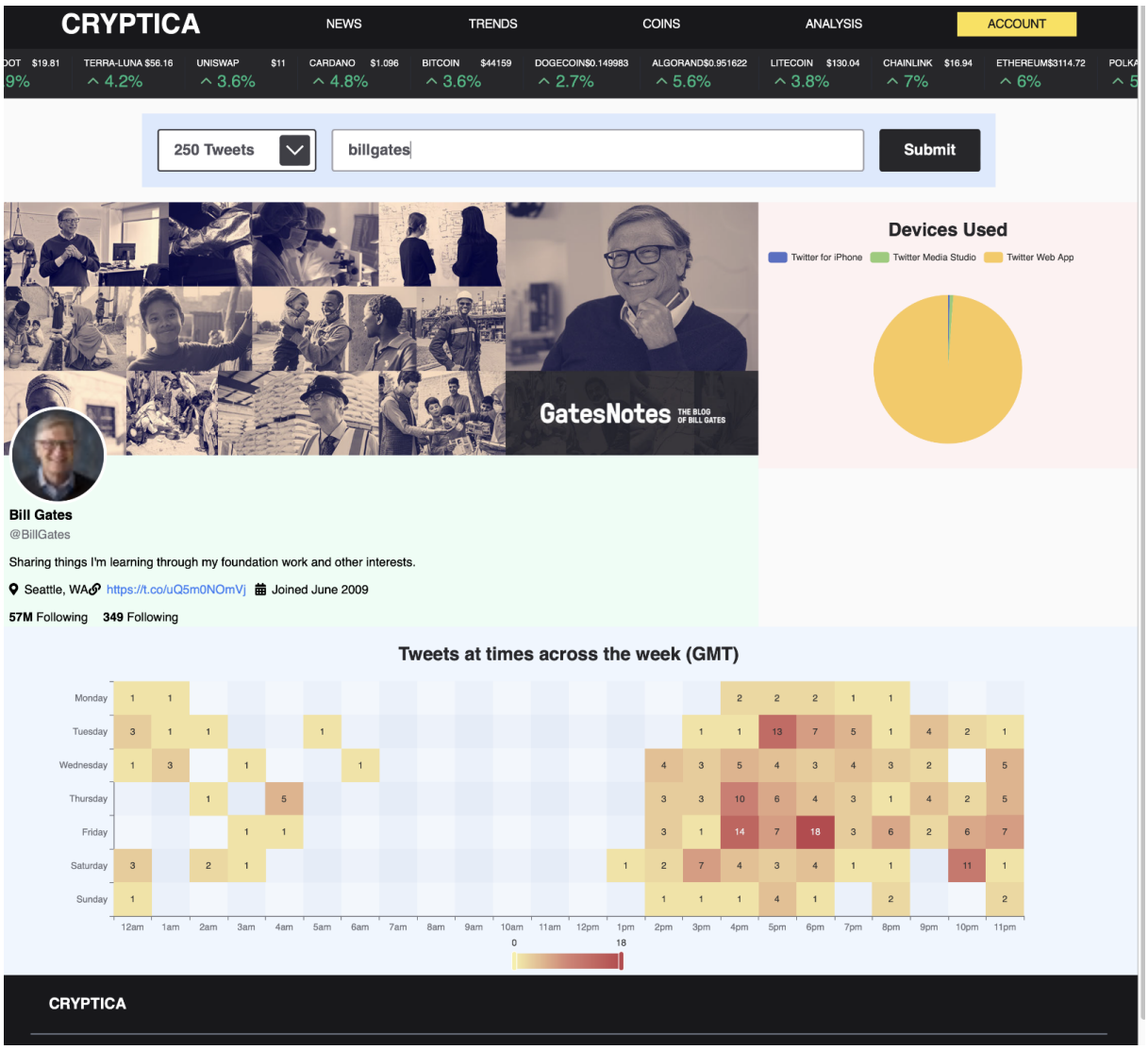

Figure 32: The analysis page after Bill Gate's Twitter account is supplied in the input field

client/pages/login/index.js

This page allows the user to login to the application

```
import { Field, Form, Formik } from 'formik';
import Link from 'next/link';
import { useState } from 'react';
import Auth from '../../services/auth';
import useUser from '../../services/user';
import { useRouter } from 'next/router';
```

```
7
8  function Login(props) {
9    const [reply, setReply] = useState(); // this creates a state
       variable for the reply from the API
10   const [button, setButton] = useState('Log In'); // this
       creates a state variable for the button text
11   const { user, loading, error } = useUser(); // this sets user,
        loading, and error from the user service
12   const Router = useRouter(); // this creates an instance of the
        router service
13   const auth = new Auth(); // this creates an instance of the
       auth service
14   if (user) { // if the user is logged in already
15     Router.push('/' + (Router.query.redirect || 'account')); //
         redirect to the account page, or the page that was
         requested before the login page
16     return <></>; // return nothing
17   }
18   if (error && error.response && error.response.status === 401)
       { // if there is an error and it is a 401 forbidden,
       display the error message
19     auth.deleteToken(); // delete the authentication token from
         local storage as it must be invalid
20   }
21   return ( // display the login form
22     <div className="container mx-auto p-4 mt-12 bg-white flex
         flex-col items-center justify-center">
23       <div className="w-10/12 sm:w-8/12 md:w-6/12 lg:w-5/12 xl:w
           -4/12 mb-4">
24         <h1 className="text-4xl font-semibold ">Welcome back.</
             h1>
25       </div>
26       <div className="w-10/12 sm:w-8/12 md:w-6/12 lg:w-5/12 xl:w
           -4/12 mb-6">
27         <Formik
28           initialValues={{ // this sets the initial values for
               the form
29             email: '',
30             password: ''
31           }}
32           onSubmit={(values, { setSubmitting }) => { // this
               sets the on submit function for the form
33             setButton('Logging in...'); // set the button text
                 to logging in
34             auth // call the auth service
```

```
35                .poster('/auth/login', { // post the login data to
                      the api
36                  email: values.email, // set the email from the
                       form
37                  password: values.password // set the password
                       from the form
38                })
39                .then((response) => {
40                  if (response.status == 200) { // if the response
                       is a 200 ok
41                    setSubmitting(false); // set the form to not
                         be submitting
42                    auth.saveToken(response.data.access_token); //
                         save the authentication token to local
                         storage
43                    Router.push('/' + (Router.query.redirect || '
                         account')); // redirect to the account page
                         , or the page that was requested before the
                         login page
44                  } else { // if the response is not a 200 ok
45                    setReply('Error: ' + response.response.data.
                         detail); // set the reply to the error
                         message
46                    setButton('Log In'); // set the button text to
                         log in
47                    setSubmitting(false); // set the form to not
                         be submitting
48                  }
49                });
50            }}
51            render={() => ( // this renders the form
52              <Form>
53                <Field
54                  id="email"
55                  name="email"
56                  placeholder="Email"
57                  type="email"
58                  className="mb-4 p-2 appearance-none block w-full
                       bg-neutral-200 placeholder-neutral-900
                       rounded border focus:border-teal-500"
59                />
60
61                <Field
62                  id="password"
63                  name="password"
```

```
                  placeholder="Password"
                  type="password"
                  className="mb-4 p-2 appearance-none block w-full bg-neutral-200 placeholder-neutral-900 rounded border focus:border-teal-500"
                />

                <div className="flex items-center">
                  <div className="w-1/2 flex items-center">
                    <a className="text-sm font-semibold text-center">
                      <p>New to Cryptica? </p>
                      <p className="hover:underline text-complementary-400">
                        <Link href="/register" className="hover:underline">
                          <p className="hover:underline text-complementary-700">Create an account</p>
                        </Link>
                      </p>
                    </a>
                  </div>
                  <button
                    className="ml-auto w-1/2 bg-neutral-800 text-white p-2 rounded font-semibold hover:bg-neutral-900"
                    type="submit"
                  >
                    {button}
                  </button>
                </div>
              </Form>
            )}
          />
        </div>
        {reply && (
          <div className="flex justify-center w-10/12 sm:w-8/12 md:w-6/12 lg:w-5/12 xl:w-4/12 bg-red-600 py-3 rounded">
            <p className="font-semibold text-white text-sm">{reply}</p>
          </div>
        )}
      </div>
    );
```

```
}

export default Login;
```

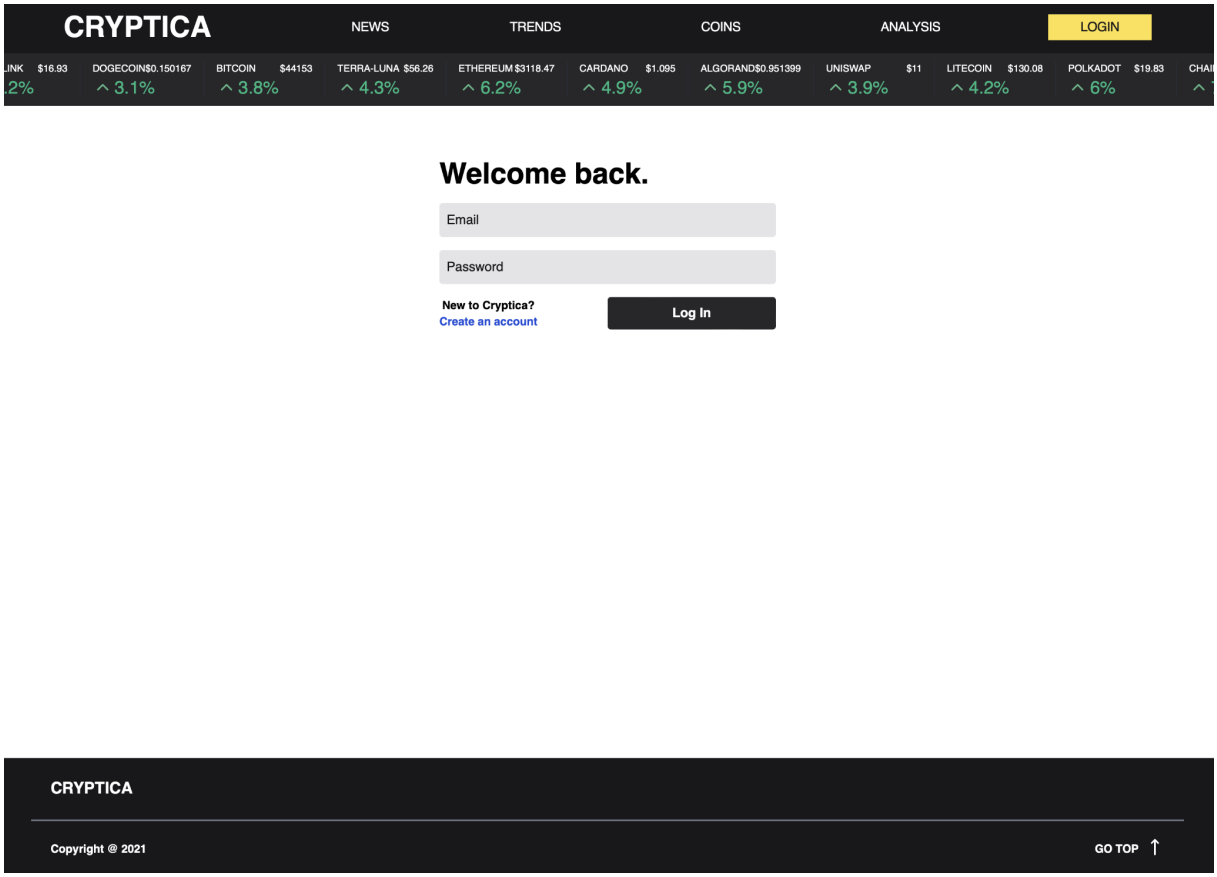

Figure 33: The Login Page

client/page/register/index.js

This page allows the user to register for an account

```
import { Field, Form, Formik } from 'formik';
import * as React from 'react';
import Router from 'next/router';
import { useState } from 'react';
import Link from 'next/link';
import Auth from '../../services/auth';
import useUser from '../../services/user';

```

```
function Register() {
  const [reply, setReply] = useState(); // this creates a state variable for the reply from the API
  const [button, setButton] = useState('Register'); // this creates a state variable for the button text
  const { user, loading, error } = useUser(); // this sets user, loading, and error from the user service
  const auth = new Auth(); // this creates an instance of the auth service
  if (user) { // if the user is logged in
    Router.push('/account'); // redirect to the account page
    return <></>;
  }
  if (error && error.response && error.response.status === 401) { // if there is an error and it is a 401 forbidden, display the error message
    auth.deleteToken(); // delete the authentication token from local storage as it must be invalid
  }

  return (
    <main className="container mx-auto p-4 mt-12 bg-white flex flex-col items-center justify-center">
      <div className="w-10/12 sm:w-8/12 md:w-6/12 lg:w-5/12 xl:w-4/12 mb-4">
        <h1 className="text-4xl font-semibold ">Welcome to Cryptica.</h1>
      </div>
      <div className="w-10/12 sm:w-8/12 md:w-6/12 lg:w-5/12 xl:w-4/12 mb-6">
        <Formik // this creates a form with initial values and on submit function
          initialValues={{
            first_name: '',
            last_name: '',
            email: '',
            password: ''
          }}
          onSubmit={(values, { setSubmitting }) => { // this sets the on submit function for the form
            setButton('Registering...');
            auth
              .poster('/auth/register', { // post the login data to the api
                first_name: values.first_name,
```

```
                  last_name: values.last_name,
                  email: values.email,
                  password: values.password
                })
                .then((response) => { //
                  if (response.status == 200) { // if the response is a 200 OK
                    setSubmitting(false); // stop the form from submitting
                    auth.saveToken(response.data.access_token); // save the authentication token returned to local storage
                    Router.push('/account');  // redirect to the account page
                  } else { // if the response is not a 200 OK
                    setReply('Error: ' + response.response.data.detail); // set the reply to the error message
                    setButton('Register'); // set the button text back to register
                    setSubmitting(false); // stop the form from submitting
                  }
                });
            }}
            render={() => ( // this renders the form
              <Form>
                <div>
                  <div className="flex">
                    <div className="w-1/2">
                      <Field
                        id="first_name"
                        name="first_name"
                        placeholder="First Name"
                        className="mb-4 py-2 pl-2 appearance-none block bg-neutral-200 placeholder-neutral-900 rounded border focus:border-teal-500"
                      />
                    </div>
                    <div className="w-1/2">
                      <Field
                        id="last_name"
                        name="last_name"
                        placeholder="Last Name"
```

```
                    className="mb-4 py-2 pl-2 appearance-none block bg-neutral-200 placeholder-neutral-900 rounded border focus:border-teal-500"
                  />
                </div>
              </div>
              <Field
                id="email"
                name="email"
                placeholder="Email"
                type="email"
                className="mb-4 p-2 appearance-none block w-full bg-neutral-200 placeholder-neutral-900 rounded border focus:border-teal-500"
              />

              <Field
                id="password"
                name="password"
                placeholder="Password"
                type="password"
                className="mb-4 p-2 appearance-none block w-full bg-neutral-200 placeholder-neutral-900 rounded border focus:border-teal-500"
              />

              <div className="flex items-center">
                <div className="w-1/2 flex items-center">
                  <a className="text-sm font-semibold text-center">
                    <p>Already got an account? </p>
                    <p className="hover:underline text-complementary-400">
                      <Link href="/login" className="hover:underline">
                        <p className="hover:underline text-complementary-700">Sign in here</p>
                      </Link>
                    </p>
                  </a>
                </div>
                <button
                  className="ml-auto w-1/2 bg-neutral-800 text-
```

```
                        white p-2 rounded font-semibold hover:bg-
                        neutral-900"
                    type="submit"
                  >
                    {button}
                  </button>
                </div>
              </Form>
            )}
          />
        </div>
        {reply && (
          <div className="flex justify-center w-10/12 sm:w-8/12 md
             :w-6/12 lg:w-5/12 xl:w-4/12 bg-red-600 py-3 rounded">
            <p className="font-semibold text-white text-sm">{reply
               }</p>
          </div>
        )}
        <div className="flex justify-center w-10/12 sm:w-8/12 md:w
           -6/12 lg:w-5/12 xl:w-4/12 py-3 rounded">
          <p className="text-sm">
            <p className="text-md text-bold">Password Requirements
               </p>
            <ul className="list-disc text-sm list-inside">
              <li>At least 8 characters</li>
              <li>At least one number</li>
              <li>At least one uppercase letter</li>
              <li>At least one special character</li>
            </ul>
          </p>
        </div>
      </main>
    );
}

export default Register;
```

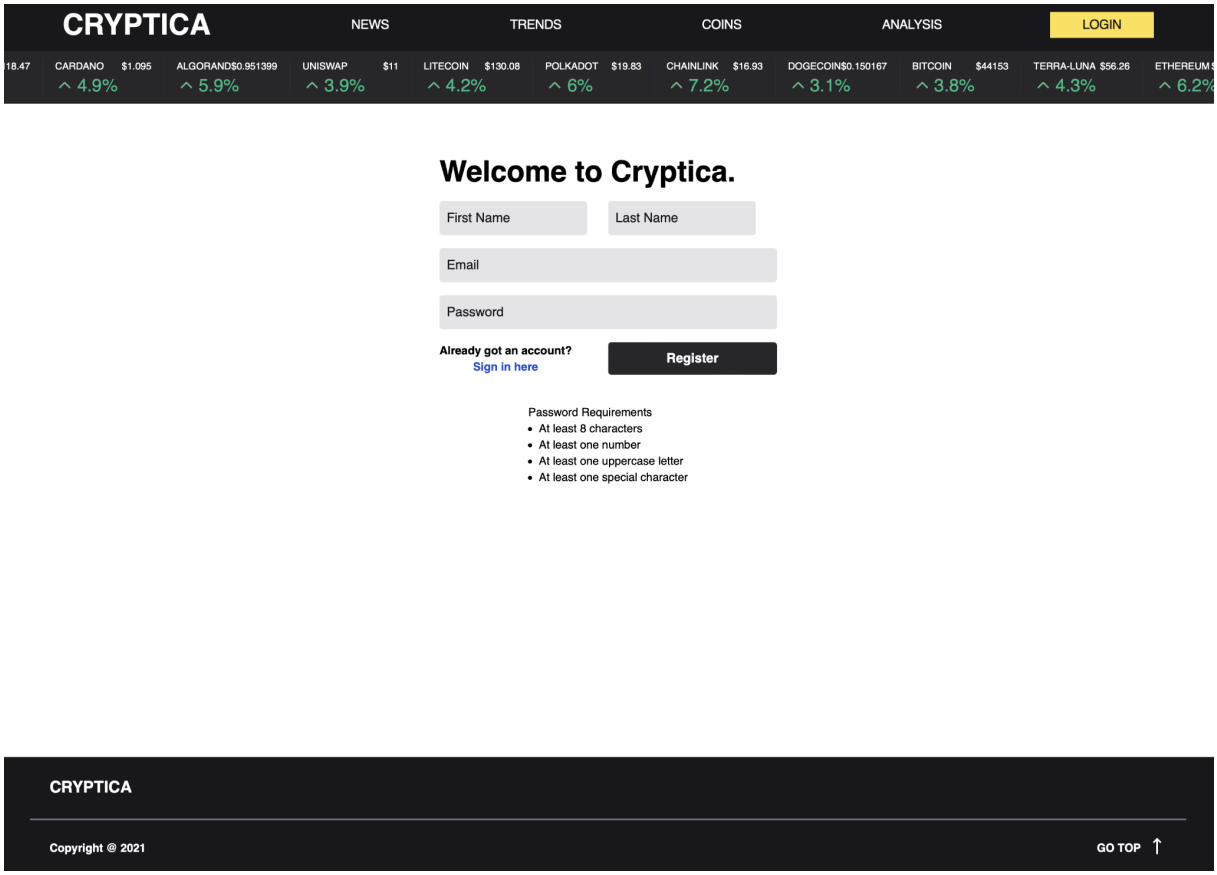

Figure 34: The register page

client/pages/news/index.js

This page shows a list of news articles from the API.

```
import Sidebar from '../../components/layout/sidebar';
import Feature from '../../components/news/feature';
import Post from '../../components/news/post';
import { useEffect, useState } from 'react';
import Auth from '../../services/auth';
import Loading from '../../components/loading';
export default function News() {
  const auth = new Auth(); // creates an instance of the Auth class
  const [data, setData] = useState(); // creates a state variable for the data
  useEffect(() => { // useEffect hook for fetching data when the
```

```
        component mounts
    auth // calls the auth instance
      .fetcher('/news/') // fetches the data from the API
      .then((res) => setData(res)) // sets the data to the state
      .catch(); // catches any errors
  }, []);

  function formatDate(date) { // function to format the date into a readable format
    const dateObj = new Date(date); // creates a new date object from the date string
    const now = new Date(); // creates a new date object for now
    const diff = now - dateObj; // calculates the difference between the two dates
    const diffDays = Math.floor(diff / (1000 * 60 * 60 * 24)); // calculates the difference in days
    if (diffDays < 1) { // if the difference is less than 1 day
      const diffHours = Math.floor(diff / (1000 * 60 * 60)); // calculates the difference in hours
      if (diffHours < 1) { // if the difference is less than 1 hour
        const diffMinutes = Math.floor(diff / (1000 * 60)); // calculates the difference in minutes
        if (diffMinutes < 1) { // if the difference is less than 1 minute
          return 'Just now';  // returns 'Just now'
        } else { // if the difference is more than 1 minute
        return `${diffMinutes} minutes ago`; // returns the difference in minutes
      }
      return `${diffHours} hours ago`; // returns the difference in hours
    }
    if (diffDays === 1) { // if the difference is 1 day
      return 'Yesterday'; // returns 'Yesterday'
    }
    if (diffDays < 7) { // if the difference is less than 7 days
      return `${diffDays} days ago`; // returns the difference in days
    } else { // if the difference is more than 7 days
      const monthNames = [ // creates an array of month names
        'January',
        'February',
        'March',
        'April',
```

```
44          'May',
45          'June',
46          'July',
47          'August',
48          'September',
49          'October',
50          'November',
51          'December'
52        ];
53        const day = dateObj.getDate(); // gets the day from the
            date object
54        const monthIndex = dateObj.getMonth(); // gets the month
            from the date object
55        const year = dateObj.getFullYear(); // gets the year from
            the date object
56        return `${monthNames[monthIndex]} ${day}, ${year}`; //
            returns the month and day
57      }
58    }
59
60    return (
61      <div className="flex bg-gray-50">
62        <div className="h-1/2 top-32 sticky w-64 hidden lg:flex">
63          <div className="flex-1 px-5">
64            <Sidebar>
65              <div className="py-5 text-6xl text-center lg:text-
                left font-medium transform">NEWS</div>
66              All our news is gathered from the internet via an
                external party, so we can{"'"}t
67              guarantee the accuracy of the news. <br />
68              For more information, visit our FAQs.
69            </Sidebar>
70          </div>
71        </div>
72
73        <div className="flex-grow ">
74          {data ? ( // if the data is set, then show the feature
              with the first data item
75            <Feature
76              id={data[0].id}
77              title={data[0].title}
78              author={data[0].publication}
79              date={formatDate(data[0].date)} // formats the date
80              image={data[0].imageurl || 'null'} // if the
                imageurl is set, then use it, otherwise use null
```

```
81            />
82          ) : (
83            <Feature />
84          )}
85
86          <div className="sm:px-2 md:px-20 border-dotted lg:border
              -l xl:border-r border-neutral-600">
87            <h2 className="py-8 text-2xl font-extrabold leading-
                tight border-b border-dotted border-neutral-600 md:
                text-4xl">
88              All Articles
89            </h2>
90            {data ? ( // if the data is set, then show the posts
                with the rest of the data items
91              data.map((post) => ( // maps the data items to the
                  post component with the data
92                <Post
93                  key={post.id}
94                  title={post.title}
95                  summary={post.description}
96                  date={formatDate(post.date)} // formats the date
97                  author={post.publication}
98                  id={post.id}
99                  image={post.imageurl}
100               />
101             ))
102           ) : ( // if the data is not set, then show a loading
                screen
103             <Post
104               key="1"
105               title="Loading..."
106               summary="Loading..."
107               date="Loading..."
108               author="Loading..."
109               id="1"
110               image=""
111             />
112           )}
113         </div>
114       </div>
115     </div>
116   );
117 }
```

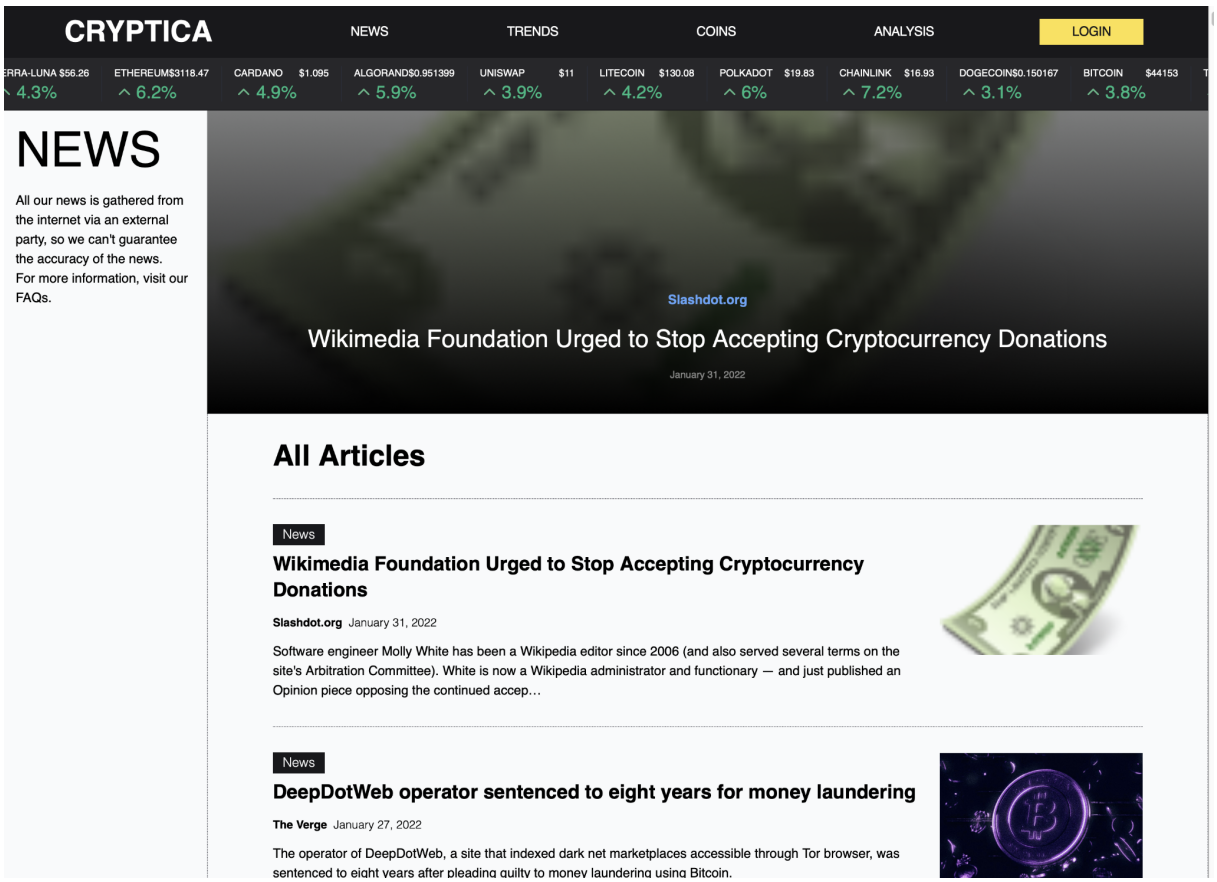

Figure 35: News index page, with the featured article at the top

client/pages/news/[id].js

This page shows details about a specific article, specified by the ID in the url. It also contains a comment box for users to write comments.

```
import { useRouter } from 'next/router';
import Content from '../../components/news/content';
import Loading from '../../components/loading';
import Auth from '../../services/auth';
import Comments from '../../components/news/comments';
import { useState, useEffect } from 'react';
const Post = (props) => {
  const auth = new Auth(); // creates an instance of the Auth class
  const router = useRouter(); // creates an instance of the router
```

```
10    const { id } = router.query; // gets the id of the news
         article from the url
11    const [data, setData] = useState(null); // creates a state
         variable for the data
12    useEffect(() => { // useEffect hook for fetching data when the
          component mounts
13      (async () => {
14        if (!id) { // if there is no id
15          return; // returns
16        }
17        const response = await auth.fetcher(`/news/${id}`, false);
              // fetches the data from the API with the id
18        setData(response); // sets the data to the state
19      })(); // calls the async function
20    }, [id]); // runs the useEffect hook when the id changes
21
22    if (!data || !id) { // if there is no data or the id is not
         set
23      return <Loading />; // returns the loading component
24    }
25
26    if (data) {
27      return (
28        <>
29          <div className="py-10 bg-primary-800 text-white pb-60">
30            <h1 className="text-4xl font-bold text-center">{data.
                title}</h1>
31            <h3 className="text-xl text-center font-medium pt-2">{
                data.date}</h3>
32          </div>
33
34          <section className="container h-96 mx-auto flex -mt-48">
35            <img className="mx-auto" src={data.imageurl} alt="
                Image" />
36          </section>
37
38          <Content content={data.content} />
39          <div className="p-5 bg-primary-800 text-white space-y-2"
             >
40            <h3 className="text-2xl text-center font-light hover:
                text-neutral-300">
41              <a href={data.url}>Read the full article here..</a>
42            </h3>
43            <h3 className="text-md text-center font-medium">
44              {data.author} | {data.publication}
```

```
45            </h3>
46          </div>
47          <Comments comments={data.comments} id={id} />
48        </>
49      );
50    }
51  };
52
53  export default Post;
```

Figure 36: An example news page, with a testing comment.

client/pages/_app.js

This file adds the layout component to every single page. The layout component contains the header and some other parts.

```
import Layout from '../components/layout/layout';

import './global.css';

function App({ Component, pageProps }) {
  return (
    <Layout> // adds the layer component around the page
      <Component {...pageProps} />
    </Layout>
  );
}

export default App;
```

client/services/auth.js

This file handles authentication and making requests and is used throughout the program.

```
import Cookies from 'universal-cookie';
const axios = require('axios');

class HTTPRequests { // creates a class for handling HTTP requests
  async fetcher(url, auth = false) { // function to make fetch GET requests
    const headers = { // headers for the request
      'Content-Type': 'application/json'
    };
    if (auth) { // if auth is true, add the JWT token to the request
      const cookies = new Cookies(); // create a new cookie object
      const token = cookies.get('token'); // get the token from the cookie
      if (!token) { // if there is no token, throw an error
        const error = new Error('Unauthorized'); // create an error
        return Promise.reject(error); // return the error
```

```
      } else { // if there is a token, add it to the request
        headers.Authorization = 'Bearer ' + token; // add the token to the request headers
      }
    }
    const res = await axios.get(`${process.env.NEXT_PUBLIC_API_URL}${url}`, { headers: headers }); // make the request
    return res.data; // return the data
  }

  async poster(url, data, auth = false) { // function to make POST requests
    const headers = {  // headers for the request
      'Content-Type': 'application/json'
    };
    if (auth) {
      const cookies = new Cookies();
      const token = cookies.get('token');
      if (!token) {
        error = new Error('No token');
        return Promise.reject(error);
      } else {
        headers.Authorization = 'Bearer ' + token;
      }
    }
    const res = await axios // make the request
      .post(`${process.env.NEXT_PUBLIC_API_URL}${url}`, data, { // send the data
        headers: headers // with the headers
      })
      .catch((error) => { // if there is an error
        return error; // return the error
      });

    return res;
  }

  async deleter(url, auth = false) { // function to make DELETE requests
    const headers = {
      'Content-Type': 'application/json'
    };
    if (auth) { //
      const cookies = new Cookies();
```

```
54        const token = cookies.get('token');
55        if (!token) {
56          error = new Error('No token');
57          return Promise.reject(error);
58        } else {
59          headers.Authorization = 'Bearer ' + token;
60        }
61      }
62      const res = await axios // make the request
63        .delete(`${process.env.NEXT_PUBLIC_API_URL}${url}`, {
64          headers: headers
65        })
66        .catch((error) => {
67          return error;
68        });
69
70      return res;
71    }
72  }
73
74  class Auth extends HTTPRequests { // function for managing
       authentication, extends the HTTPRequests class and inherits
       its methods
75    saveToken(token) { // function to save the token
76      const cookies = new Cookies(); // create a new cookie object
77      cookies.set('token', token, { // set the token in the cookie
78        path: '/',
79        expires: new Date(Date.now() + (1000 * 60 * 60 * 24 * 7)),
             // set the expiry date to 1 week from now
80        sameSite: true // set the cookie to only be accessible
             from the same site
81      });
82      return Promise.resolve(); // return a resolved promise
83    }
84
85    deleteToken() { // function to delete the token
86      const cookies = new Cookies(); // create a new cookie object
87      cookies.remove('token', { path: '/' }); // remove the token
         from the cookie
88      return; // return nothing
89    }
90    useUser() { // function to check if the user is logged in
91      const { data, error } = useSWR(['/auth/me', true], this.
         fetcher); // get the user data from the API
92      return { // return the data and error
```

```
      user: data, // the user data
      loading: !error && !data, // if there is an error or no data, the user is loading
      error: error // the error
  }}

}
export default Auth; // export the class
```

component/comments.js

Component for a comment. Allows the user to delete the comment using the `auth.deleter` function, if the comment is their own.

```
import { useRouter } from 'next/router';
import Link from 'next/link';
import Auth from '../services/auth';

const Users = (props) => {
  const router = useRouter();
  const auth = new Auth();
  async function deleteComment(comment_id) {
    if (!comment_id) {
      return;
    }
    await auth.deleter(`/news/0/comments/${comment_id}`, true).catch((err) => {
      console.log(err.message);
    });
    router.reload();
  }
  return (
    <>
      {props.comments.map((comment, key) => (
        <div
          key={key}
          className="flex items-center py-4 mx-auto border border-black sm:px-8 md:px-12 sm:py-4 w-full md:w-full px-3 mb-2 mt-8"
        >
          <div>
            <h3 className='className="text-lg font-bold text-primary-800 sm:text-xl md:text-2xl'>
```

```
                <Link href={`/news/${comment.news_id}`}>{comment.title}</Link>
              </h3>
              <p className="text-sm font-bold text-neutral-600">{comment.date}</p>
              <p className="mt-2 text-base sm:text-lg md:text-normal">{comment.content}</p>
              {(props.user == comment.user_id || props.user.admin) && (
                <button
                  onClick={() => deleteComment(comment.id)}
                  className="text-sm font-bold text-neutral-600"
                >
                  Delete Comment
                </button>
              )}
            </div>
          </div>
        ))}
      </>
    );
};

export default Users;
```

component/loading.js

Component that indicates that the page is loading.

```
const Loading = (props) => {
  return (
    <>
      <div className="flex justify-center items-center h-full -mt-24">
        <div
          className={
            'animate-spin rounded-full h-32 w-32 border-b-2 ' +
            (props.dark ? 'border-neutral-100' : 'border-neutral-900')
          }
        >
        </div>
      </div>
```

```
    </>
  );
};

export default Loading;
```

component/layout/layout.js

Component that provides the basic page layout that is applied to every page.

```
import NavBar from './navbar/navbar';
import Head from 'next/head';
const Layout = (props) => {
  return (
    <>
      <Head>
        <title>CRYPTICA</title>
        <meta name="viewport" content="viewport-fit=cover, width=device-width, initial-scale=1.0" />
      </Head>
      <div className="flex flex-col min-h-screen">
        <NavBar />
        <main className="flex-grow relative flex-1 dark:bg-black dark:text-white bg-white">
          {props.children}
        </main>
        {/* Footer */}
        <section className="h-full bg-primary-900">
          <div className="py-6 px-16 flex justify-between">
            <div>
              <h1 className="font-bold text-white text-xl">CRYPTICA</h1>
            </div>
          </div>

          <div className="border-t-2 mx-10 border-gray-500"></div>

          <div className="py-4 py-6 px-16 flex justify-between">
            <div>
              <h1 className="font-semibold text-white text-sm">Copyright @ 2021</h1>
            </div>
```

```

            <div>
              <a href="#" className="flex space-x-2 text-white
                 hover:text-yellow-400">
                <p className="font-semibold text-sm">GO TOP</p>
                <svg
                  xmlns="http://www.w3.org/2000/svg"
                  className="h-6 w-6  -mt-1"
                  fill="none"
                  viewBox="0 0 24 24"
                  stroke="currentColor"
                >
                  <path
                    strokeLinecap="round"
                    strokeLinejoin="round"
                    strokeWidth="2"
                    d="M8 7l4-4m0 0l4 4m-4-4v18"
                  />
                </svg>
              </a>
            </div>
          </div>
        </section>
      </div>
    </>
  );
};

export default Layout;
```

component/layout/navbar/ticker.js

Component that fetches data from a public cryptocurrency price API, and displays the price change and price in a ticker format.

```

import Price from './price';
import { useEffect, useState } from 'react';
import axios from 'axios';
import FinancialTicker from '../ticker';
import TickerList from '../ticker/general';

const Ticker = () => {
```

```
9   const round = (num) => {
10    return Math.round(num * 10000) / 10000;
11  };
12
13  const coins = [
14    'bitcoin',
15    'ethereum',
16    'cardano',
17    'binance-coin',
18    'xrp',
19    'dogecoin',
20    'polkadot',
21    'solano',
22    'uniswap',
23    'litecoin',
24    'terra-luna',
25    'chainlink',
26    'algorand'
27  ];
28
29  const url =
30    'https://api.coingecko.com/api/v3/simple/price?ids=' +
31    coins.join('%2c') +
32    '&vs_currencies=usd&include_24hr_change=true';
33
34  const [data, setData] = useState([]);
35
36  const [visible, setVisible] = useState(true);
37
38  useEffect(() => {
39    axios
40      .get(url)
41      .then((response) => {
42        setData(response.data);
43        //console.log(response.data);
44        return;
45      })
46      .catch((error) => {
47        return;
48      });
49  }, []);
50  return (
51    <>
52      {visible && (
53        <nav className="bg-primary-800 nav flex flex-wrap items-
```

```
              center justify-between overflow-x-auto border-b
              border-primary-900">
            <TickerList>
              {Object.keys(data).map((coin, key) => {
                return (
                  <FinancialTicker
                    key={key}
                    id={key}
                    symbol={coin}
                    lastPrice={data[coin].usd}
                    percentage={Math.abs(Math.round(data[coin].
                       usd_24h_change * 10) / 10)}
                    currentPrice={round(data[coin].usd)}
                    positive={data[coin].usd_24h_change > 0}
                  />
                );
              })}
            </TickerList>
          </nav>
        )}
      </>
    );
  };

  export default Ticker;
```

component/layout/account.js

Component that displays either account, or login, depending on whether the user is logged in or not.

```
import Auth from '../../../services/auth';
import useUser from '../../../services/user';
const Account = () => {
  const { user, loading, error } = useUser();
  const auth = new Auth();
  if (user) {
    return <>ACCOUNT</>;
  }
  if (error && error.response && error.response.status === 401)
     {
    auth.deleteToken();
    return <>LOGIN</>;
```

```
  }
  return <>LOGIN</>;
};

export default Account;
```

component/coin/graph.js

Component that fetches price data from an API and displays it in a candlestick chart for a specified cryptocurrency.

```
import { useState, useEffect } from 'react';
import axios from 'axios';
import ReactECharts from 'echarts-for-react';

const Graph = (props) => {
  const [data, setData] = useState();
  const formateDate = (timestamp) => {
    const d = new Date(timestamp);
    const month = [
      'Jan',
      'Feb',
      'Mar',
      'Apr',
      'May',
      'Jun',
      'Jul',
      'Aug',
      'Sep',
      'Oct',
      'Nov',
      'Dec'
    ][d.getMonth()];
    const day = d.getDate();
    const year = d.getFullYear();
    return `${day} ${month} ${year}`;
  };

  useEffect(async () => {
    if (!props.coin) {
      setData([]);
      return;
    }
```

```
33      const result = await axios(
34        `https://api.coingecko.com/api/v3/coins/${props.coin.
            toLowerCase()}/market_chart?vs_currency=usd&days=365`
35      ).catch(() => {
36        setData([]);
37        return;
38      });
39      //console.log(result.data);
40      result && result.data && setData(result.data.prices);
41    }, [props.coin]);
42
43    if (!data) {
44      return null;
45    }
46
47    const option = {
48      xAxis: {
49        type: 'category',
50        data: data.map((d) => formateDate(d[0]))
51      },
52      yAxis: {
53        type: 'value'
54      },
55      series: [
56        {
57          data: data.map((d) => d[1]),
58          type: 'line',
59          showSymbol: false
60        }
61      ],
62
63      title: {
64        text: props.coin + '/usd',
65        x: 'center',
66        top: '10px'
67      },
68      tooltip: {
69        trigger: 'axis',
70        axisPointer: {
71          type: 'cross',
72          label: {
73            backgroundColor: '#6a7985'
74          }
75        }
76      }
```

```
  };

  return (
    <>
      <ReactECharts
        option={option}
        notMerge={true}
        lazyUpdate={true}
        style={{ height: '100%', width: '100%' }}
      />
    </>
  );
};

export default Graph;
```

component/analysis/tweet.js

Component that imitates a tweet embed, that can be supplied with data to appear like a tweet.

```
import Image from 'next/image';

export default function Tweet(props) {
  const authorUrl = `https://twitter.com/${props.tweet.username}`;
  const likeUrl = `https://twitter.com/intent/like?tweet_id=${props.tweet.id}`;
  const retweetUrl = `https://twitter.com/intent/retweet?tweet_id=${props.tweet.id}`;
  const replyUrl = `https://twitter.com/intent/tweet?in_reply_to=${props.tweet.id}`;
  const tweetUrl = `https://twitter.com/${props.tweet.username}/status/${props.tweet.id}`;

  const formattedText = props.tweet.tweet.replace(/https:\/\/[\n\S]+/g, '');

  return (
    <>
      <div className="flex items-center">
        <a className="flex h-12 w-12" href={authorUrl} target="_blank" rel="noopener noreferrer">
```

```
            <Image
              alt={props.tweet.username}
              height={48}
              width={48}
              src={props.user.avatar}
              className="rounded-full"
            />
          </a>
          <a
            href={authorUrl}
            target="_blank"
            rel="noopener noreferrer"
            className="author flex flex-col ml-4 !no-underline"
          >
            <span
              className="flex items-center font-bold !text-neutral-900 dark:!text-neutral-100 leading-5"
              title={props.tweet.username}
            >
              {props.tweet.name}
              {props.user.is_verified ? (
                <svg
                  aria-label="Verified Account"
                  className="ml-1 text-complementary-500 dark:text-white inline h-4 w-4"
                  viewBox="0 0 24 24"
                >
                  <g fill="currentColor">
                    <path d="M22.5 12.5c0-1.58-.875-2.95-2.148-3.6.154-.435.238-.905.238-1.4 0-2.21-1.71-3.998-3.818-3.998-.470-.92.084-1.336.25C14.818 2.415 13.51 1.512 1.5s-2.816.917-3.437 2.25c-.415-.165-.866-.25-1.336-.25-2.11 0-3.818 1.79-3.818 4 0 .494.083.964.237 1.4-1.272.65-2.147 2.018-2.147 3.6 0 1.495.782 2.798 1.942 3.486-.02.17-.032.34-.032.514 0 2.21 1.708 4 3.818 4 .47 0 .92-.086 1.335-.25.62 1.334 1.926 2.25 3.437 2.25 1.512 0 2.818-.916 3.437-2.25.415.163.865.248 1.336.248 2.11 0 3.818-1.79 3.818-4 0-.174-.012-.344-.033-.513 1.158-.687 1.943-1.99 1.943-3.484zm-6.616-3.334l-4.334 6.5c-.145.217-.382.334-.625.334-.143
```

```
                        0-.288-.04-.416-.126l-.115-.094-2.415-2.415
                        c-.293-.293-.293-.768 0-1.06s.768-.294 1.06
                         0l1.77 1.767 3.825-5.74c.23-.345.696-.436
                        1.04-.207.346.23.44.696.21 1.04z" />
                  </g>
                </svg>
              ) : null}
            </span>
            <span className="!text-neutral-500" title={`@${props.
               tweet.username}`}>
              @{props.tweet.username}
            </span>
          </a>
          <a className="ml-auto" href={authorUrl} target="_blank"
             rel="noopener noreferrer">
            <svg viewBox="328 355 335 276" height="24" width="24"
               xmlns="http://www.w3.org/2000/svg">
              <path
                d="M 630, 425    A 195, 195 0 0 1 331, 600    A
                   142, 142 0 0 0 428, 570    A  70,  70 0 0 1
                   370, 523    A  70,  70 0 0 0 401, 521    A  70,
                     70 0 0 1 344, 455    A  70,  70 0 0 0 372,
                   460    A  70,  70 0 0 1 354, 370    A 195, 195
                   0 0 0 495, 442    A  67,  67 0 0 1 611, 380
                   A 117, 117 0 0 0 654, 363    A  65,  65 0 0 1
                   623, 401    A 117, 117 0 0 0 662, 390    A  65,
                     65 0 0 1 630, 425    Z"
                style={{ fill: '#3BA9EE' }}
              />
            </svg>
          </a>
        </div>
        <div className="mt-4 mb-1 leading-normal whitespace-pre-
           wrap text-lg !text-neutral-700 dark:!text-neutral-300">
          {formattedText}
        </div>
        {props.tweet.photos && props.tweet.photos.length ? (
          <div
            className={
              props.tweet.photos.length === 1
                ? 'inline-grid grid-cols-1 gap-x-2 gap-y-2 my-2'
                : 'inline-grid grid-cols-2 gap-x-2 gap-y-2 my-2'
            }
          >
            {props.tweet.photos.map((m, key) => (
```

```
72              <div key={key}>
73                <img alt={props.tweet.tweet} src={m} size="small"
                     className="rounded" />
74              </div>
75            ))}
76          </div>
77        ) : null}
78        <a
79          className="!text-neutral-500 text-sm hover:!underline"
80          href={tweetUrl}
81          target="_blank"
82          rel="noopener noreferrer"
83        >
84          <time title={`Time Posted: ${props.tweet.datetime}`}
               dateTime={props.tweet.datetime}>
85            {props.tweet.datetime}
86          </time>
87        </a>
88        <div className="flex !text-neutral-700 dark:!text-neutral
             -300 mt-2">
89          <a
90            className="flex items-center mr-4 !text-neutral-500
                hover:!text-complementary-600 transition hover:!
                underline"
91            href={replyUrl}
92            target="_blank"
93            rel="noopener noreferrer"
94          >
95            <svg className="mr-2" width="24" height="24" viewBox="
                0 0 24 24">
96              <path
97                className="fill-current"
98                d="M14.046 2.242l-4.148-.01h-.002c-4.374 0-7.8
                     3.427-7.8 7.802 0 4.098 3.186 7.206 7.465 7.37
                     v3.828c0
                     .108.045.286.12.403.143.225.385.347.633.347.138
                      0 .277-.038.402-.118.264-.168 6.473-4.14
                     8.088-5.506 1.902-1.61 3.04-3.97 3.043-6.312v
                     -.017c-.006-4.368-3.43-7.788-7.8-7.79zm3.787
                     12.972c-1.134.96-4.862 3.405-6.772 4.643V16.67
                     c0-.414-.334-.75-.75-.75h-.395c-3.66
                     0-6.318-2.476-6.318-5.886 0-3.534 2.768-6.302
                     6.3-6.302l4.147.01h.002c3.532 0 6.3 2.766 6.302
                      6.296-.003 1.91-.942 3.844-2.514 5.176z"
99              />
```

```
            </svg>
            <span>{new Number(props.tweet.replies_count).toLocaleString()}</span>
          </a>
          <a
            className="flex items-center mr-4 !text-neutral-500 hover:!text-green-600 transition hover:!underline"
            href={retweetUrl}
            target="_blank"
            rel="noopener noreferrer"
          >
            <svg className="mr-2" width="24" height="24" viewBox="0 0 24 24">
              <path
                className="fill-current"
                d="M23.77 15.67c-.292-.293-.767-.293-1.06 0l-2.22 2.22V7.65c0-2.068-1.683-3.75-3.75-3.75h-5.85c-.414 0-.75.336-.75.75s.336.75.75.75h5.85c1.24 0 2.25 1.01 2.25 2.25v10.24l-2.22-2.22c-.293-.293-.768-.293-1.06 0s-.294.768 0 1.06l3.5 3.5c.145.147.337.22.53.22s.383-.072.53-.22l3.5-3.5c.294-.292.294-.767 0-1.06zm-10.66 3.28H7.26c-1.24 0-2.25-1.01-2.25-2.25V6.46l2.22 2.22c.148.147.34.22.532.22s.384-.073.53-.22c.293-.293.293-.768 0-1.06l-3.5-3.5c-.293-.294-.768-.294-1.06 0l-3.5 3.5c-.294.292-.294.767 0 1.06s.767.293 1.06 0l2.22-2.22V16.7c0 2.068 1.683 3.75 3.75 3.75h5.85c.414 0 .75-.336.75-.75s-.337-.75-.75-.75z"
              />
            </svg>
            <span>{new Number(props.tweet.retweets_count).toLocaleString()}</span>
          </a>
          <a
            className="flex items-center !text-neutral-500 hover:!text-red-600 transition hover:!underline"
            href={likeUrl}
            target="_blank"
            rel="noopener noreferrer"
          >
            <svg className="mr-2" width="24" height="24" viewBox="0 0 24 24">
              <path
                className="fill-current"
```

```
                d="M12 21.638h-.014C9.403 21.59 1.95 14.856 1.95
                   8.478c0-3.064 2.525-5.754 5.403-5.754 2.29 0
                   3.83 1.58 4.646 2.73.813-1.148 2.353-2.73
                   4.644-2.73 2.88 0 5.404 2.69 5.404 5.755 0
                   6.375-7.454 13.11-10.037 13.156H12zM7.354 4.225
                   c-2.08 0-3.903 1.988-3.903 4.255 0 5.74 7.035
                   11.596 8.55 11.658 1.52-.062 8.55-5.917
                   8.55-11.658
                   0-2.267-1.822-4.255-3.902-4.255-2.528 0-3.94
                   2.936-3.952 2.965-.23.562-1.156.562-1.387
                   0-.015-.03-1.426-2.965-3.955-2.965z"
              />
            </svg>
            <span>{new Number(props.tweet.likes_count).
               toLocaleString()}</span>
          </a>
        </div>
      </>
    );
  }
```

components/analysis/search.js

Search bar component.

```
const Search = () => {
  return (
    <div className="space-x-5 flex">
      <div className="relative inline-flex self-center flex-
         initial">
        <svg
          className="text-white bg-primary-700 absolute top-0
             right-0 m-2 pointer-events-none p-2 rounded"
          xmlns="http://www.w3.org/2000/svg"
          width="40px"
          height="40px"
          viewBox="0 0 38 22"
          version="1.1"
        >
          <g stroke="none" strokeWidth="1" fill="none" fillRule=
             "evenodd">
            <g transform="translate(-539.000000, -199.000000)"
               fill="#ffffff" fillRule="nonzero">
```

```
                <g id="Icon-/-ArrowRight-Copy-2" transform="translate(538.000000, 183.521208)">
                  <polygon
                    id="Path-Copy"
                    transform="translate(20.000000, 18.384776) rotate(135.000000) translate(-20.000000, -18.384776) "
                    points="33 5.38477631 33 31.3847763 29 31.3847763 28.999 9.38379168 7 9.38477631 7 5.38477631"
                  />
                </g>
              </g>
            </g>
          </svg>
          <select className="text-xl font-bold rounded border-2 border-primary-700 text-neutral-600 h-14 w-44 pl-5 pr-10 bg-white hover:border-neutral-400 focus:outline-none appearance-none">
            <option>Bitcoin</option>
            <option>Ethereum</option>
            <option>Doge</option>
            <option>Litecoin</option>
            <option>Cardano</option>
          </select>
        </div>
        <input
          placeholder="elonmusk"
          className="flex-auto text-xl font-bold rounded border-2 border-primary-700 text-neutral-600 h-14 pl-5 pr-10 bg-white hover:border-neutral-400 focus:outline-none appearance-none"
        />
        <button className="text-xl font-bold rounded text-white h-14 px-8 bg-primary-800 hover:bg-primary-900 focus:outline-none appearance-none">
          Submit
        </button>
      </div>
    );
};

export default Search;
```

components/analysis/ohcl.js

Another graph component, that takes in data and displays it as a candlestick chart.

```
import ReactECharts from 'echarts-for-react';

const OHCL = (props) => {
  const data = props.data;
  //console.log(data);
  const formatInt = (int) => {
    if (int.slice(-1) === '0') {
      int = int.slice(0, -1);
      return formatInt(int);
    } else {
      return parseFloat(int, 10);
    }
  };

  if (!data) {
    return null;
  }

  const downColour = '#ec0000';
  const downBorderColour = '#8A0000';
  const upColour = '#00da3c';
  const upBorderColour = '#008F28';
  const option = {
    dataset: {
      source: data
    },

    tooltip: {
      trigger: 'axis',
      axisPointer: {
        type: 'cross'
      },
      renderMode: 'html',
      padding: 4,
      /*formatter: function (params) {
        console.log(params[0]);
        const colour = params[0].data[1] > params[0].data[4] ? 'red' : 'green';
        return `<a className="text-xl text-${colour}-500">$${params[0].data[4]}</a>`;
      }*/
```

```
40        extraCssText: '@apply bg-neutral-200 p-10'
41      },
42      grid: [
43        {
44          left: '8%',
45          right: '5%',
46          bottom: '10%'
47        }
48      ],
49      xAxis: [
50        {
51          type: 'time',
52          scale: true
53        }
54      ],
55      yAxis: [
56        {
57          scale: true,
58          type: 'value',
59          axisLabel: {
60            formatter: '${value}'
61          }
62        }
63      ],
64      dataZoom: [
65        {
66          type: 'inside',
67          xAxisIndex: [0, 1]
68        }
69      ],
70
71      series: [
72        {
73          type: 'candlestick',
74          itemStyle: {
75            color: upColour,
76            color0: downColour,
77            borderColor: upBorderColour,
78            borderColor0: downBorderColour
79          },
80          encode: {
81            x: 0,
82            y: [1, 4, 3, 2],
83            tooltip: [1, 4, 3, 2]
84          },
```

```
85          markLine: {
86            itemStyle: {
87              color: 'rgba(255, 173, 177, 1)'
88            },
89            symbol: 'none',
90            lineStyle: {
91              type: 'solid',
92              capp: 'round',
93              width: 2,
94              opacity: 0.8,
95              color: 'blue'
96            },
97            label: {
98              show: true,
99              fontSize: 17,
100             formatter: 'Tweet',
101             color: 'blue',
102             fontWeight: 'bold',
103             opacity: 0.8
104           },
105           data: [
106             {
107               name: 'Tweet',
108               xAxis: data[29][0] - 30000,
109               type: 'max'
110             }
111           ]
112         }
113       }
114     ]
115   };
116   return (
117     <>
118       <ReactECharts
119         option={option}
120         notMerge={true}
121         lazyUpdate={true}
122         style={{ height: '100%', width: '100%' }}
123       />
124     </>
125   );
126 };
127
128 export default OHCL;
```

TESTING

CLIENT APPLICATION TESTING

I have tested the clients performance by using Google's lighthouse page analysis tool.

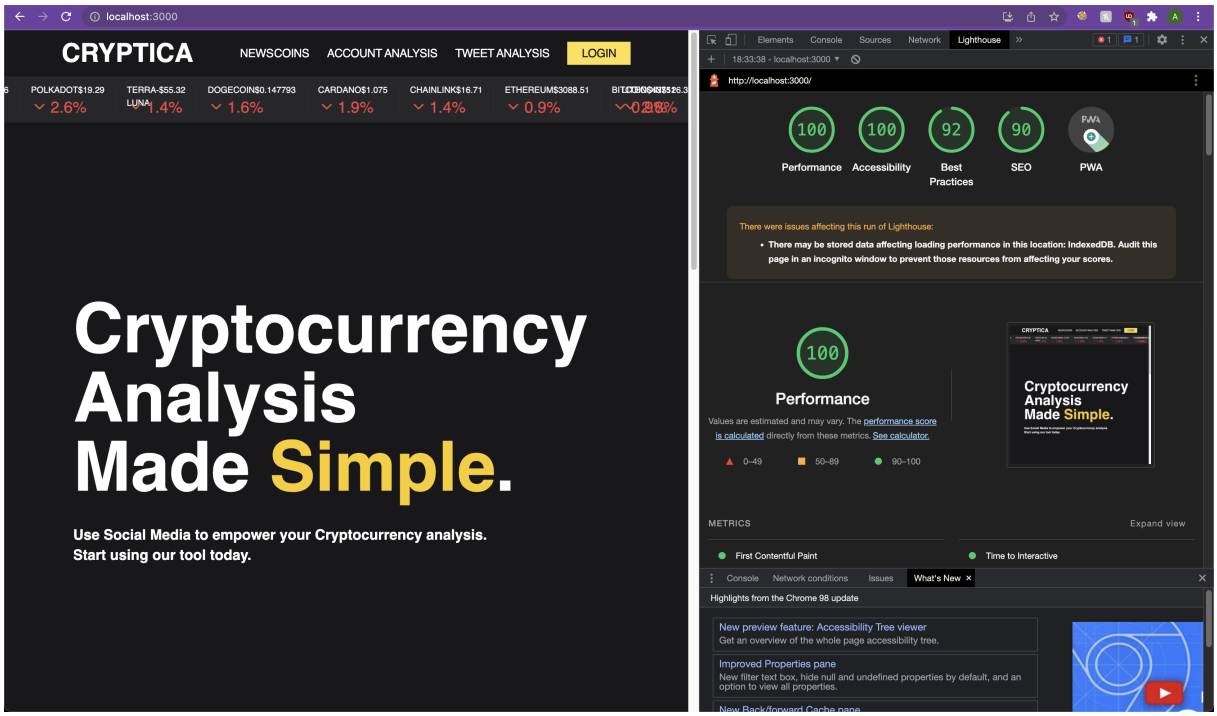

Figure 37: Screenshot of the lighthouse tool

It performed excellently, scoring 90-100 in all 4 categories.

To test the clients functionality, I prepared a table full of tests. Then I recorded a video where I go through and check each test. You can see the video and the table below.

Video Link: https://youtu.be/zSj_id9I7-c

ID	Component	Test	Result	Video Times-tamp
1.1	News index page	There should be a news page that displays all the articles in the database	Pass	0:04
1.2	News index page	There should be a feature article, that has a full size image and headline at the top of the page	Pass	0:03
1.3	News index page	There should be a list of articles with title, image, date, and other info	Pass	0:04
1.4	News index page	Each article on news index page is clickable, and clicking brings you to the articles page	Pass	0:13
2.1	News article page	Each article should have its own page that is accessible from the news index page	Pass	0:13
2.2	News article page	There should be a full size article image and article title displayed	Pass	0:13
2.3	News index page	On each articles page, an excerpt from the begininning of the article is displayed	Pass	0:13
2.4	News index page	Each article page has a link to the original article	Pass	0:20
2.5	News index page	Each article page has a comments field at the bottum. This should be grayed out if the user is not logged in	Pass	0:19

ID	Component	Test	Result	Video Times-tamp
3.1	Coin index page	There should be a coin index page that has a table with information about the top 50 cryptocurrencies	Pass	0:25
3.2	Coin index page	Statistics about each coin should be displayed in the table	Pass	0:25
3.3	Coin index page	Each coin in the table should be clickable, and should bring you to the coins unique page	Pass	0:40
4.1	Individual coin page	Each coin should have a page that displays some information about the coin	Pass	0:40
4.2	Individual coin page	Each coins page should have a graph showing the price of the coin against the dollar in the last year	Pass	0:42
4.3	Individual coin page	Other metrics such as market cap should be shown on the coins page	Pass	0:40
4.4	Individual coin page	Related news articles should be displayed on the coins page	Pass	0:40
4.5	Individual coin page	Clicking on one of the related articles should take you to the articles page	Pass	0:56
4.6	Individual coin page	Each coin page should have a description of the coin	Pass	0:40

ID	Component	Test	Result	Video Times-tamp
5.1	Registration page	There should be a registration page, that has a form for name, email, and password	Pass	1:09
5.2	Registration page	Attempting to register with an email that already exists should display an error	Pass	1:26
5.3	Registration page	Attempting to register with a password that does not meet the security requirements should display an error	Pass	1:20
5.4	Registration page	Registering successfully should redirect you to the account page	Pass	1:37
6.1	Account page	There should be an account page that is only accessible to logged in users	Pass	1:37
6.2	Account page	There should be a banner that welcomes the name of the user	Pass	1:37
6.3	Account page	The logged in users email should be displayed	Pass	1:37
6.4	Account page	There should be a logout button, that upon clicking logs the user out	Pass	1:48
7.1	Login page	There should be a login page that allows existing users to login	Pass	1:49
7.2	Login page	There should be a form that users can enter their email and password	Pass	1:49

ID	Component	Test	Result	Video Times-tamp
7.3	Login page	Users attempting to login with invalid credentials should be shown an error message	Pass	1:59
7.4	Login page	Users logging in with a correct password should be redirected to the account page	Pass	2:06
8.1	Account analysis page	There should be an account analysis page	Pass	2:10
8.2	Account analysis page	On the page there should be a form that has a username input, and a dropdown allowing a quantity of tweets to be selected	Pass	2:10
8.3	Account analysis page	After submitting the form, graphs should populate the page	Pass	2:27
8.4	Account analysis page	There should be a pie chart graph showing the distribution of devices that the user has used to tweet from	Pass	2:27
8.5	Account analysis page	There should be a heatmap showing the times that the user has historically tweeted at	Pass	2:30
8.6	Account analysis page	There should be a box displaying how many followers the user has, in addition to other details such as the profile picture and following count	Pass	2:27

ID	Component	Test	Result	Video Times-tamp
9.1	Tweet analysis page	There should be an individual tweet analysis page	Pass	3:08
9.2	Tweet analysis page	There should be a form that has a field for entering a username, and a select box for choosing between one of several cryptocurrencies to search for	Pass	3:08
9.3	Tweet analysis page	Once submitted, a list of relevent tweets should be displayed on the right side of the page	Pass	3:16
9.4	Tweet analysis page	The user should be able to click on and select a tweet. Doing so should reveal a new section to the page	Pass	3:30
9.5	Tweet analysis page	This section should contain a graph that shows the selected coins price at the time of the tweet. The time of the tweet should be highlighted on the graph, showing the impact that the tweet has had.	Pass	3:30
9.6	Tweet analysis page	The predicted sentiment of the tweet should be displayed in this section	Pass	3:38

ID	Component	Test	Result	Video Times-tamp
10.1	Ticker layout	There should be a rotating ticker below the menu bar the displays the live price of certain coins	Pass	4:18
10.2	Ticker layout	Clicking on any of the coins in the ticker should bring you to the coins page	Pass	4:21
11.1	Authentication	Only logged in users should be able to access either of the analysis pages	Pass	4:57
11.2	Authentication	Only logged in users should be able to comment on an article	Pass	4:32
11.3	Authentication	Only logged in users should be able to view a users profile	Pass	5:11
11.4	Authentication	A button in the menu bar should display either “ACCOUNT” or “LOG IN” depending on whether the user is logged in or not	Pass	1:46

SERVER CODE TESTING

You can find in the below table the set of tests that I will be performing on some individual functions and part of my code. Below the table, you will find evidence of each the tests.

Component	Test	Expected Result	Test Data
RSA	Test to generate an RSA key	The RSA key generator should produce a public and private keypair	The keysize will be set to 8 bits. This is so the numbers are low and easy to verify manually.
RSA	Test to check RSA key is valid by encrypting plaintext using RSA algorithm	The RSA public key should be capable of being used to encrypt a value	The plaintext we will use is 12345
RSA	Test to check RSA key is valid by decryption cipher using RSA algorithm	The RSA private key should be able to decrypt the cipher back to the original plaintext value using the formula	The ciphertext we will use will be the result of the previous step
Miller Rabin Primality	Test to check the Miller Rabin function can identify whether a number is prime	True should be returned for each of the inputs	[100, 291, 949, 3107, 3615, 3693, 6381, 7869, 7913] - known non primes
Miller Rabin Primality	Test to check the Miller Rabin function can identify whether a number is non prime	False should be returned for each of the inputs	[89, 857, 2473, 4273, 6029, 6791, 7789, 7823, 7901, 7919] - known primes
Sentiment Analysis Model	Test positive input to check whether the model identifies the input as positive	The model should output a number between 60% and 100% to classify an item as positive	The input “Bitcoin is so cool! I think it is great” will be inputted into the model
Sentiment Analysis Model	Test negative input to check whether the model identifies the input as negative	The model should output a number between 0% and 40% to classify an item as negative	The input “I think Cryptocurrencies are so stupid and a waste of resources” will be inputted into the model

Component	Test	Expected Result	Test Data
Sentiment Analysis Model	Check accuracy of model on training and testing data	As outlined in the objectives, ideally above 75% accuracy	As outlined in the design phase, I will use part of the training dataset that has been split off to test and evaluate the model
JWT Creation	Test to generate a JSON Web Token with supplied user data	JSON Web Token Created that contains encoded data, signed with RSA private key	The following JSON should be used: `{'user': test', 'email': 'test@test.com'}`
JWT Verification	Test to verify JSON Web Token using a valid signed token	JSON Web Token should be initialised into class object, which can then be used to decode the token to view it's data	Token from previous test should be used as input
JWT Verification	Invalid signed RSA	The class should throw an error stating that the signature is invalid	Token from previous step with a modified signature should be used
Base64 Encoding	Check that ascii text can be base64 encoded	The test data input should be returned encoded using base64. The validity of this can be verified using a number of online base64 encoding tools	"hello"
Base64 Decoding	Check that base64 can decode to ascii text	The base64 should be decoded and returned as ASCII	aGVsbG8= (base64 of hello)

You can find how I've done each test and evidence for each one below.

RSA Testing

To test my RSA function works as intended, I will be testing it by generating very small RSA keys which I can manually verify using the maths and algorithms I have described earlier. I am using a keysize of 8 to test, which means that n is of a maximum size of $(2^8)^2$, or 65536.

```
# greatest common divisor
def gcd(a, b):
    return egcd(a, b)[0] # return the gcd of a and b

keysize = 8

p = generate_prime(keysize) # generate a prime of keysize 8
q = generate_prime(keysize) # generate a prime of keysize 8

n = p * q # n is the product of p and q

while True: # generate value of e until we get one that is coprime with n
    e = random.randint(2 ** (keysize - 1), 2 ** (keysize)) # generate a random numb
    if gcd(e, (p - 1) * (q - 1)) == 1: # if the gcd of e and (p - 1) * (q - 1) is 1
        break # break the loop

g, x, y = egcd(e, (p - 1) * (q - 1)) # use the extended euclidean algorithm to find
d = x % ((p - 1) * (q - 1)) # d is the inverse of e mod (p - 1) * (q - 1)


publickey = (n, e) # public key is n and e
privatekey = (n, d) # private key is n and d

print("public key:", publickey) # print the public key
print("private key:", privatekey) # print the private key
```

PROBLEMS OUTPUT DEBUG CONSOLE GITLENS TERMINAL

```
❯ python rsa.py
public key: (42558, 151)
private key: (42558, 26791)
```

Figure 38: Screenshot of the test and generated keys

My program generated the pair 42588, 151 for the public key, and 42558, 26791 for the private key.

Using the formula $C = P^e \, mod \, n$, where C is the cipher text, and P is the plaintext, I am going to attempt to encrypt the plaintext of 12345. Substituting the values into the formula gets the following result:

$C = 12345^{151} \, mod \, 42558$
$C = 27351$

```
>>> (12345 ** 151) % 42558
27351
```

Figure 39: Screenshot of encryption calculation

Here we have calculated the cipher text to be 27351. Now to decrypt, we will use the formula $P = C^d \, mod \, n$.

$P = 27351^{26791} \, mod \, 42558$
$P = 12345$

```
>>> (27351 ** 26791) % 42558
12345
```

Figure 40: Screenshot of decryption calculation

We have now successfully proved that encryption and decryption works using RSA, as we have got our original plaintext back.

Miller_Rabin Function Testing

The Miller_Rabin function should return True when a number is prime, and false when a number is not prime. To test it, I supplied it with a list of known primes and non primes and checked to ensure that the outputted result matched the expected result.

```python
nonprimes = [100, 291, 949, 3107, 3615, 3693, 6381, 7869, 7913]
primes = [89, 857, 2473, 4273, 6029, 6791, 7789, 7823, 7901, 7919]

for x in nonprimes:
    print(x, miller_rabin(x))

for x in primes:
    print(x, miller_rabin(x))
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE GITLENS TERMINAL

```
❯ python tool.py
100 False
291 False
949 False
3107 False
3615 False
3693 False
6381 False
7869 False
7913 False
89 True
857 True
2473 True
4273 True
6029 True
6791 True
7789 True
7823 True
7901 True
7919 True
```

Figure 41: Screenshot of prime test

As you can see above, the test passed and the function correctly identified all the numbers as prime or non prime.

Sentiment Analysis Model Testing

The function `predict_sentiment` outputs a number from 0 to 1 depending on the predicted sentiment of the input. To calculate a percentage, the number should be times by 100. As mentioned above, a value below 40% is considered negative, and above 60% positive.

First, I have tested the input `Bitcoin is so cool! I think it is great` with my model. It successfully managed to predict the sentiment as positive, with a value of 75.99%.

```
predict_sentiment(model, "Bitcoin is so cool! I think it is great")
0.7598732499697987
```

Figure 42: Predicting sentiment of a Positive Input

Next, I entered a negative input into my `predict_sentiment` function. I used the input `I think Cryptocurrencies are so stupid and a waste of resources`.

```
predict_sentiment(model, "I think Cryptocurrencies are so stupid and a waste of resources")
0.3845724089059668
```

Figure 43: Predicting sentiment of a Negative Input

This time, the model predicted the negative text had a sentiment value of 38.46%. This is close, but just below the 40% border, suggesting that the model could be improved further.

Finally, I evaluated the model using the built in evaluation tool. This tests it against the designated test data which was split off from the original dataset.

```
testloss, testacc = evaluate(model, iterator)

print(f'Test Loss: {testloss}, Test Acc: {testacc}%')

Test Loss: 0.292, Test Acc: 87.96%
```

Figure 44: Evaluating the Sentiment Analysis Model

The model achieved a test accuracy of 87.96%, with a loss of 0.292 The loss is a useful measure of how well a model is performing. It is calculated based on training and validation data, and is a summation of errors made for each example in the sets. It is used when optimising models. My loss value and accuracy suggest that my model performs ok, but could be improved with fine tuning and further training. For my purposes I consider my model successful.

JSON Web Token

Firstly, using the `AccessToken` class I defined in `/api/core/security.py`, I attempted to initialise the class using the JSON data `{'user': 'test', 'email': 'test@test.com'}`.

```
>>>
>>>
>>> from core.security import AccessToken as AccessToken
>>>
>>> token = AccessToken(data={'user': 'test', 'email': 'test@test.com'})
>>>
>>> token
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidGVzdCIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImV4cCI6MTY0OTg3Mzg5OC40NDkxMjN9.YMt-l_NynxnkCHvtq8_LZFpge
XC1zrIevAhEc8p-z3zDCftn09weaoXNQ2zEy1VPPkWcv4vrn5SSmsOLG4cTDt2PMGTI-PrJ8ql0DEiGdNr7m7m5l5yX5mWrlhXdQBKyaiRhFOtQA1hL9wwR4CmCT_boDW249_0ou1fGDTMlwy_C
rsDK0x5dICNtCydmN-1VkpcW9rdtjOL-N6YlQHJmARfqXa_k_jPwtRYCJ4SG4LyLZAlBKZg2ZQWlCRFZd9drqZpje_EOmfm9ZRjchJTZQ54thof1_vAM8sJDVdCYxIxzVS-vcIe6lkb2mtXV2qV
ymnW2HjZ4VttqkEIqUAQHGw
>>>
>>> token.decode_token()
{'user': 'test', 'email': 'test@test.com', 'exp': 1649873898.449123}
>>>
```

Figure 45: Generating a JSON Web Token using the AccessToken class

My class successfully created a JSON Web Token signed with my RSA private key. Using the online JSON Web Debugging tool https://jwt.io, I was able to verify that the signature and JSON Web Token was valid by inputting the token and my RSA public key.

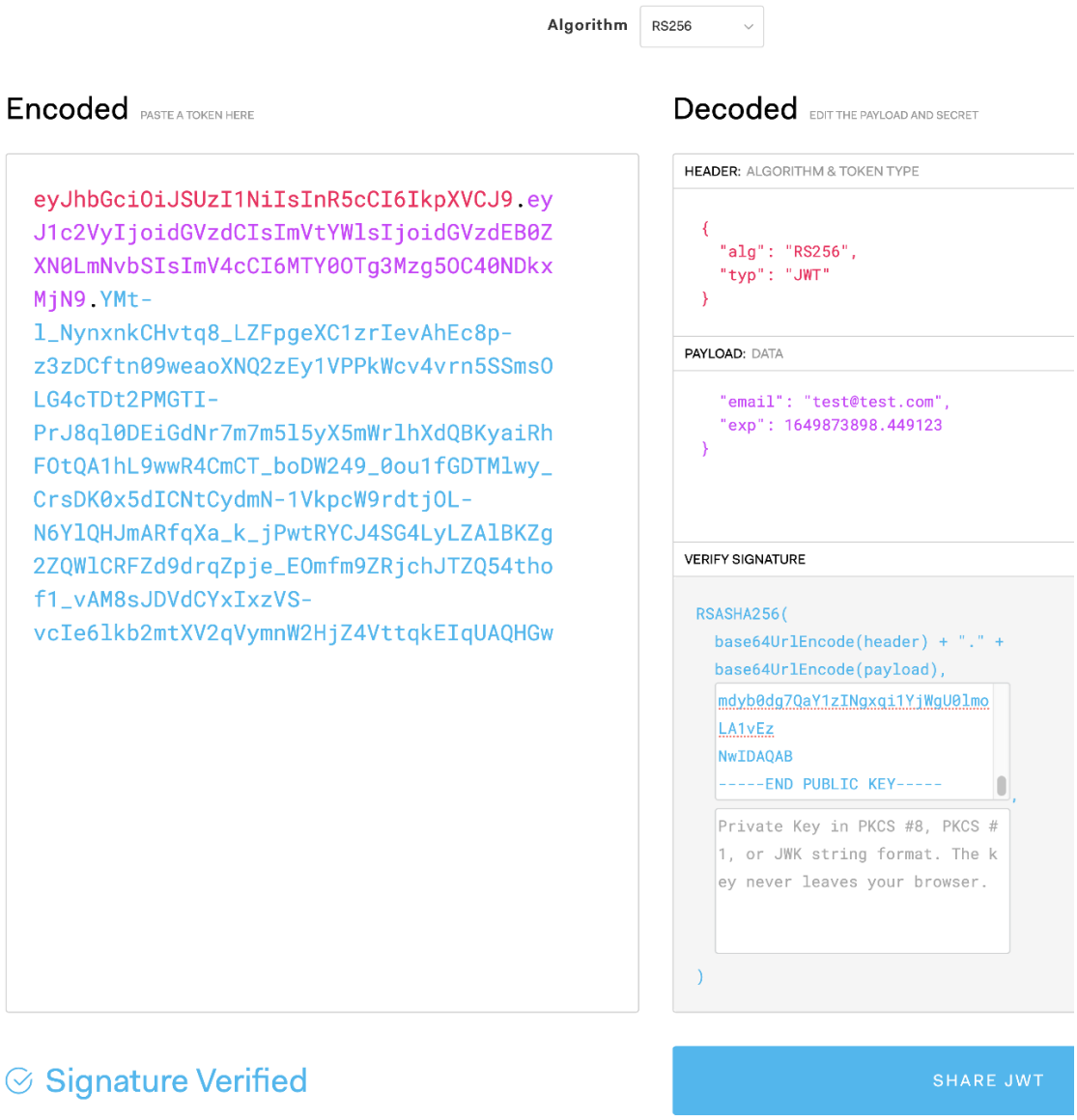

Figure 46: Verifying the created JSON Web Token

Next, I attempted to initialise the AccessToken class using the JSON Web Token just created in the previous step.

```
>>> token = AccessToken(token='eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidGVzdCIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImV4cCI6MTY0OTg3Mzg5OC40NDk
xMjN9.YMt-l_NynxnkCHvtq8_LZFpgeXC1zrIevAhEc8p-z3zDCftn09weaoXNQ2zEy1VPPkWcv4vrn5SSmsOLG4cTDt2PMGTI-PrJ8ql0DEiGdNr7m7m5l5yX5mWrlhXdQBKyaiRhFOtQA1hL9
wwR4CmCT_boDW249_0ou1fGDTMlwy_CrsDK0x5dICNtCydmN-1VkpcW9rdtjOL-N6YlQHJmARfqXa_k_jPwtRYCJ4SG4LyLZAlBKZg2ZQWlCRFZd9drqZpje_EOmfm9ZRjchJTZQ54thof1_vAM
8sJDVdCYxIxzVS-vcIe6lkb2mtXV2qVymnW2HjZ4VttqkEIqUAQHGw')
>>>
>>> token.decode_token()
{'user': 'test', 'email': 'test@test.com', 'exp': 1649873898.449123}
>>>
>>> token.verify_token()
True
>>>
```

Figure 47: The token being verified using the AccessToken class

The class successfully verified the token using the RSA public key, and successfully decoded it to get the original data inputted, proving the class works as intended.

Finally, I attempted to use the class with an invalid JSON Web Token. By changing the very last character of the token created in the previous steps, the signature becomes invalid. When attempting to initiate the class using this modified token, an error was thrown, proving that only tokens with valid signatures can be verified and used.

```
>>> token = AccessToken(token='eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidGVzdCIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImV4cCI6MTY0OTg3Mzg5OC40NDk
xMjN9.YMt-l_NynxnkCHvtq8_LZFpgeXC1zrIevAhEc8p-z3zDCftn09weaoXNQ2zEy1VPPkWcv4vrn5SSmsOLG4cTDt2PMGTI-PrJ8ql0DEiGdNr7m7m5l5yX5mWrlhXdQBKyaiRhFOtQA1hL9
wwR4CmCT_boDW249_0ou1fGDTMlwy_CrsDK0x5dICNtCydmN-1VkpcW9rdtjOL-N6YlQHJmARfqXa_k_jPwtRYCJ4SG4LyLZAlBKZg2ZQWlCRFZd9drqZpje_EOmfm9ZRjchJTZQ54thof1_vAM
8sJDVdCYxIxzVS-vcIe6lkb2mtXV2qVymnW2HjZ4VttqkEIqUAQHGb')
[('rsa routines', 'RSA_padding_check_PKCS1_type_1', 'invalid padding'), ('rsa routines', 'rsa_ossl_public_decrypt', 'padding check failed')]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/arthur/Code/crypticaapi/core/security.py", line 45, in __init__
    raise Exception("Invalid token") # if not raise an exception
Exception: Invalid token
>>>
```

Figure 48: Token with invalid signature being rejected

Base64

Using the `Base64` class, I encoded the string 'hello', and got the output `aGVsbG8=`. This matches the expected value, which can be verified online using a number of tools.

Then, I decoded the same string and got back the original input of 'hello'. This proves the test successful, and that the base64 encoder and decoder works as intended.

```
>>> from utils.base64 import Base64
>>>
>>> base64 = Base64()
>>>
>>> base64.encode('hello')
'aGVsbG8='
>>>
>>> base64.decode('aGVsbG8=')
'hello'
>>>
```

Figure 49: Base64 class encoding and decoding hello.

API TESTING

Testing my API server requires me to manually send multiple tests with different types of data to each of the routes. I have organised my testing with the below table, which contains a summary of what the test and expected result is. The table has been orientated horizontally to fit on the page.

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
1.1	/auth/login	Login using valid credentials	POST	email: testing@email.com, password: Testing123!	User credentials matched against values in database using hashing. JSON Web Token created using user details, and signed using RSA private key	JSON Web Token	Pass
1.2	/auth/login	Login using invalid credentials	POST	email: testing@email.com, password: InvalidPassword	User credentials do not match database	401 Unauthorised Error	Pass
1.3	/auth/login	Login using no credentials	POST	email: testing@email.com, password: None	Lack of details, unable to check database for match	401 Unauthorised Error	Pass
2.1	/auth/me	Request with valid authentication	GET	authentication: user / admin	JSON Web Token signature verified using RSA Public Key. User data from token exists in database	200 OK	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
2.2	/auth/me	Request with invalid / no authentication	GET	authentication: invalid / none	JSON Web Token signature is not valid	401 Unauthorised Error	Pass
3.1	/auth/register	Registration with valid data, no existing account	POST	name: valid_name, email: valid_email, password: valid_password	Supplied user details are inserted into database. JSON Web Token created and signed using supplied values and RSA private key	JSON Web Token	Pass
3.2	/auth/register	Registration with valid data, already existing account	POST	name: valid_name, email: valid_email, password: valid_password	Supplied user details already exist in database, error thrown	409 Conflict Error	Pass
3.3	/auth/register	Registration with invalid / null data	POST	name: invalid_name / none, email: invalid_email / none, password: invalid_password / none	Details do not meet requirements, throw error	422 Invalid Error	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
4.1	/crypto/{TICKER}/{TIME}	Requesting cryptocurrency data with valid data	GET	ticker: BTCUSDT, time: 1640995200	Binance API called from the server, data from supplied time until an hour later is fetched and returned	Price data returned in OHCL format	Pass
4.2	/crypto/{TICKER}/{TIME}	Requesting cryptocurrency data with invalid / null data	GET	ticker: invalid_ticker / none, time: invalid_time / none	Binance API throws an error which is handled	400 Error	Pass
5.1	/news	Requesting list of news	GET	None	News fetched from database using query	List of news	Pass
5.2	/news	Creating a new news article entry as an authenticated admin	POST	authentication: admin	User admin status is checked, then news inserted into database	200 OK, ID of new article	Pass
5.3	/news	Creating a new news article entry as a user / unauthenticated	POST	authentication: user / invalid / none	User admin status is invalid, error thrown	401 Unauthorised Error	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
6.1	/news/{ID}	Requesting a valid ID article info	GET	id: valid_id	Specified news id is selected from the database and returned	Full details about article	Pass
6.2	/news/{ID}	Requesting an invalid / non existing ID article info	GET	id: invalid_id	Specified news id can not be found in the database, nothing returned	404 Not found Error	Pass
7.1	/news/{ID}/comments	Requesting list of comments on valid article	GET	id: valid_id	Specified news id is selected from the database and comments returned using join statement	List of comments	Pass
7.2	/news/{ID}/comments	Requesting list of comments on invalid article	GET	id: invalid_id	Specified news id can not be found in the database, nothing returned	404 Not found Error	Pass
7.3	/news/{ID}/comments	Creating a new comment on a valid article as an authenticated user	POST	id: valid_id, authentication: user / admin, comment: comment content	Comment inserted into comments table using supplied data	200 OK	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
7.4	/news/{ID}/comments	Creating a new comment on a valid article as an unauthenticated user	POST	id: valid_id, authentication: none / invalid, comment: comment content	Authentication found be invalid, error thrown	401 Unauthorised Error	Pass
8.1	/news/{ID}/comments/{COMMENT_ID}	Authenticated user deleting their own comment	DELETE	authentication: comment_author, id: valid_id, comment_id: valid_comment_id	Authentication checked that it matches the comment author ID, comment dropped from database	200 OK	Pass
8.2	/news/{ID}/comments/{COMMENT_ID}	Authenticated user deleting another users comment	DELETE	authentication: non_comment_author, id: valid_id, comment_id: valid_comment_id	Authentication checked and ID does not match the author ID	401 Unauthorised Error	Pass
8.3	/news/{ID}/comments/{COMMENT_ID}	Authenticated admin deleting another users comment	DELETE	authentication: admin, id: valid_id, comment_id: valid_comment_id	Authentication checked to be of admin status, comment dropped	200 OK	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
9.1	/news/comments	Authenticated admin requesting all comments	GET	authentication: admin	Authentication is a valid admin, all users selected from database where admin = true	All comments in database	Pass
9.2	/news/comments	User / unauthenticated user requesting all comments	GET	authentication: user / none	Authentication is not admin status, error thrown	401 Unauthorised Error	Pass
10.1	/news/search	Search request for a specified phrase	POST	search_term: term	Database searched using complex select query, all matching entries returned	All articles that have content matching term, if any	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
11.1	/twitter/search	Authenticated user searching for tweets matching specified arguments	POST	authentication: user, search_term: term, user: username, count: number	Authentication valid, twitter API is searched using specified query arguments. If any matching results found, returned	All tweets found from twitter API matching arguments, if any	Pass
11.2	/twitter/search	Unathenticated user searching for tweets	POST	authentication: none	Authentication invalid, error thrown	401 Unauthorised Error	Pass
12.1	/users	Authenticated admin requesting all users	GET	authentication: admin	Authentication is admin status, all users selected and returned from database	List of all users from database returned	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
12.2	/users	Authenticated admin creating new user	POST	authentication: admin, user: new_user_data	Authentication is admin status, user is inserted into database if it does not already exist	New User ID	Pass
13.1	/users/{ID}	Authenticated admin requesting user info	GET	authentication: admin, id: valid_user_id	Authentication is admin status, user is selected and returned from the database	User Details	Pass
13.2	/users/{ID}	Authenticated admin modifying user info	PUT	authentication: admin, id: valid_user_iduser: new_user_data	Authentication is admin status, user is modified in the database with the new supplied details	New User Details	Pass
13.3	/users/{ID}	Authenticated admin deleting a user	DELETE	authentication: admin, id: valid_user_id	Authentication is admin status, specified user is dropped from the table	200 OK	Pass

ID	API Endpoint	Description	HTTP Method	Test Data	Server Processing	Expected API Response	Pass/Fail
14.1	/users/{ID}/profile	Authenticated user requesting a user's profile	GET	authentication: user, id: valid_user_id	User is authenticated. Query runs to select some basic user information from the user table, and a list of the user's comments and the articles that the comments are from using inner join. This is returned as a JSON object	User Details and Comments	Pass
15.1	/users/admin	Authenticated admin requesting a list of administrator accounts	GET	authentication: admin	Authentication is admin status, all users from database who are admin are selected and returned	List of admins	Pass
16.1	/users/count	Authenticated admin requesting a count of all users	GET	authentication: admin	Authentication is admin status, using count aggregate SQL function, number of users are returned	Number of users	Pass

Evidence

I have included evidence for all of the above tests. You can find the evidence by matching the ID in the table with the corresponding test below. The evidence comes in two parts, the request and response. The request is the data sent to the API server, in the form of a HTTP request. The response is what the server replies with. I have used the HTTP Testing tool Postman to test, and the screenshots below are all from that. Some of the responses are long and as a result are not all visible in the screenshots. Some of the requests also contain variables. `{{baseUrl}}` refers to the URL that the API server is accessible from. In my case it was running locally, so the URL was `http://localhost:8000`. `[USER_TOKEN]` and `[ADMIN_TOKEN]` refer to two JSON Web Tokens that my server has created that can authenticate either a user or an admin. In the case of these tests, it means that when either feature in the headers of the request, that the user is authenticated.

1.1 Request

```
POST /api/auth/login HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Content-Length: 67

{
    "email": "testing@email.com",
    "password": "Testing123!"
}
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJzdWIiOiJ0ZXN0aW5nQGVtYWlsLmNvbSIsInBlcm1pc3Npb25zIjoidXNlciIsImV4cCI6MTY0OTg2MDY5My43NTIwODN9.
        jKx0VpY-4SkkiEA97FnmUOmi92lOMGkdpNxbMwWpgMBRY7q-lqrYAJsNRRBAPOOt8XgR6-iQmqYwtB73iWVzlThfMJCqS-Iisc8Hlj5aby7UCyHnWRk7KOJRwX0cI2FvHajF
        5qyRjw0ljx9REeG9TWKD8ul1NX6aplqHg-auBlXF2VX2wWYTL2nJuK4PQiAWrUpusLSc5WMTncyl2pTHsLT6u1eEhOowZ_Y45Bu6vwETQAdwCHwtJSP5yu8dQhteLk5IfzzP
        a1JjnCpJ8FvBvpk6LHIAz5HJheYiSvpMk69QJxj6yDHE0E7fcqTa4JYBqYUvt7Bb_80XBSCqweztxQ",
    "token_type": "bearer"
}
```


1.2 Request

```
POST /api/auth/login HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Content-Length: 71

{
    "email": "testing@email.com",
    "password": "InvalidPassword"
}
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON

```
{
    "detail": "Incorrect username or password"
}
```

1.3 Request

```
POST /api/auth/login HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Content-Length: 39

{
    "email": "",
    "password": ""
}
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON

```
{
    "detail": "Incorrect username or password"
}
```

2.1 Request

```
GET /api/auth/me HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [USER_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
{
    "status": 200
}
```

2.2 Request

```
GET /api/auth/me HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON

```
{
    "detail": "Failed to validate credentials"
}
```

3.1 Request

```
POST /api/auth/register HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Content-Length: 137

{
  "email": "testingemail@aqa.org.uk",
  "password": "SecurePassword123!",
```

```
10    "first_name": "testing",
11    "last_name": "testing example"
12  }
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1  {
2      "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
           eyJzdWIiOiJ0ZXN0aW5nZW1haWxAYXFhLm9yZy51ayIsInBlcm1pc3Npb25zIjoidXNlciIsImV4c
           CI6MTY0OTg2MTA3OS44OTMwNjF9.
           YgDy5nqjFpVLTB8CJOIZ9ckvXOFxBToEBifDVZeOviIH53c7s78liNnkoApYXTBaEU7HBS_DQQ6N_
           eiyS-yuRSteYGHNvTxSW4RLq_o52fGo5lhsSW6Z9rC8P44tDzZ9MezXsq-MOGtytglTFSP5HsF2z3
           0Z_1qpSGrLrPXNYYi0EnQ2seSol3lyZnfmsfCW-5rTaaSxA4z6L0KzxKmgINAsk2xsRQbXGh5aGUk
           AC81_xuLVHB7wEUmHBKCOPLp8aKlWmpI-BIMo10d3KxtV6bsFuNkZp_wGgOWb3NsubQVcTz_Witpf
           cmGMzMoCXdxrpo-BZSoJc9qZhFKf2uwOqA",
3      "token_type": "bearer"
4  }
```

3.2 Request

```
1  POST /api/auth/register HTTP/1.1
2  Host: {{baseUrl}}
3  Content-Type: application/json
4  Accept: application/json
5  Content-Length: 127
6
7  {
8    "email": "test@test.com",
9    "password": "SecurePassword123!",
10   "first_name": "testing",
11   "last_name": "testing example"
12 }
```

Response

Body Headers (5) Status Code 409 Conflict

Pretty Raw Preview JSON

```
1 {
2     "detail": "Account already exists"
3 }
```

3.3 Request

```
1  POST /api/auth/register HTTP/1.1
2  Host: {{baseUrl}}
3  Content-Type: application/json
4  Accept: application/json
5  Content-Length: 74
6
7  {
8    "email": "",
9    "password": "",
10   "first_name": "",
11   "last_name": ""
12 }
```

Response

Body Headers (4) Status Code 400 Bad Request

Pretty Raw Preview JSON

```
1 {
2     "detail": "Invalid Password"
3 }
```

4.1 Request

```
1 GET /api/crypto/BTCUSDT/1633046400 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    [
        1633048200000,
        "43872.99000000",
        "43913.21000000",
        "43870.12000000",
        "43904.47000000",
        "29.68011000",
        1633048259999,
        "1302600.95548760",
        844,
        "16.15721000",
        "709073.51545090",
        "0"
```


4.2 Request

```
GET /api/crypto/invalid/1633046400 HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
{
    "status_code": 400,
    "detail": "Invalid Ticker",
    "headers": null
}
```


5.1 Request

```
GET /api/news/ HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    {
        "id": 785,
        "title": "Crypto exchange Binance wins dismissal of U.S. lawsuit over digital token sales - Reuters.com",
        "publication": "Reuters",
        "imageurl": "https://www.reuters.com/resizer/o_hXSPxMRIlY15gFdDhX5OPDxhQ=/728x381/smart/filters:quality(80)/cloudfront-us-east-2.images.arcpublishing.com/reuters/TNJPSXZUUVJZPAM3V4QLJCSXWI.jpg",
        "description": "A federal judge on Thursday dismissed a lawsuit accusing Binance, the world's largest cryptocurrency exchange by trading volume, of violating U.S. securities laws by selling unregistered tokens and failing to register as an exchange or broker-dealer.",
        "date": "2022-03-31T19:07:00Z"
```


5.2 Request

```
POST /api/news HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Authorization: Bearer [ADMIN_TOKEN]
Content-Length: 185

{
    "publication": "test",
    "author": "test",
    "title": "test",
    "description": "test",
    "content": "test",
    "url": "test",
    "imageUrl": "test",
    "date": "test"
}
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 {
2     "id": 813
3 }
```

5.3 Request

```
1  POST /api/news HTTP/1.1
2  Host: {{baseUrl}}
3  Content-Type: application/json
4  Accept: application/json
5  Content-Length: 185
6
7  {
8      "publication": "test",
9      "author": "test",
10     "title": "test",
11     "description": "test",
12     "content": "test",
13     "url": "test",
14     "imageUrl": "test",
15     "date": "test"
16 }
```

Response

Body Headers (4) Status Code 403 Forbidden

Pretty Raw Preview JSON

```
1 {
2     "detail": "The user doesn't have enough privileges"
3 }
```

6.1 Request

```
1 GET /api/news/100 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
1 {
2     "title": "Solana, rival of Ethereum, is already the seventh most valuable
          cryptocurrency on the market, surpassed Dogecoin!",
3     "content": "This Tuesday, the 'altcoin' Solana (SOL) became the seventh
          cryptocurrency with the highest market capitalization above Dogecoin , Elon
          Musk's favorite. This currency, rival of Ethereum , has grown m… [+4452
          chars]",
4     "author": "Mairem Del Río",
5     "publication": "Entrepreneur",
6     "imageurl": "https://assets.entrepreneur.com/content/3x2/2000/
          1631057048-Sep8Solanacriptomonedaaltcoin.jpg",
7     "url": "https://www.entrepreneur.com/article/384229",
8     "date": "2021-09-08T11:00:00Z"
9 }
```


6.2 Request

```
1 GET /api/news/100000 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 null
```

7.1 Request

```
1 GET /api/news/814/comments HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 [
2     {
3         "id": 31,
4         "user_id": 92,
5         "news_id": 814,
6         "content": "testing comment",
7         "date": "2022-04-06 16:03:44"
8     }
9 ]
```

7.2 Request

```
1 GET /api/news/999/comments HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 null
```

7.3 Request

```
1 POST /api/news/814/comments HTTP/1.1
2 Host: {{baseUrl}}
3 Content-Type: application/json
4 Accept: application/json
5 Authorization: Bearer [USER_TOKEN]
6 Content-Length: 36
7
8 {
9     "content": "testing comment"
10 }
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 {
2     "user_id": 92,
3     "news_id": 814,
4     "date": "2022-04-06 16:03:44",
5     "content": "testing comment"
6 }
```

7.4 Request

```
1 POST /api/news/814/comments HTTP/1.1
2 Host: {{baseUrl}}
3 Content-Type: application/json
4 Accept: application/json
5 Content-Length: 36
6
7 {
8     "content": "testing comment"
9 }
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON

```
1 {
2     "detail": "Failed to validate credentials"
3 }
```

8.1 Request

```
1 DELETE /api/news/814/comments/31 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
4 Authorization: Bearer [USER_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1  31
```

8.2 Request

```
1 DELETE /api/news/814/comments/33 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
4 Authorization: Bearer [USER_TOKEN]
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON

```
1 {
2     "detail": "Unauthorized"
3 }
```

8.3 Request

```
1 DELETE /api/news/814/comments/32 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
4 Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1  32
```

9.1 Request

```
GET /api/news/comments/ HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    {
        "id": 33,
        "user_id": 94,
        "news_id": 814,
        "content": "testing comment",
        "date": "2022-04-06 16:06:55"
    },
    {
        "id": 30,
        "user_id": 90,
        "news_id": 785,
        "content": "",
        "date": "2022-04-02 17:43:40"
```


9.2 Request

```
GET /api/news/comments/ HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON


```
{
    "detail": "Failed to validate credentials"
}
```


10.1 Request

```
POST /api/news/search HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Content-Length: 27

{
    "phrase": "bitcoin"
}
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    {
        "title": "Bitcoin holds ground after touching highest this year - Reuters",
        "imageurl": "https://www.reuters.com/resizer/aZbDTblfA7PvYaZb68U0gy-DZRk=/1200x628/smart/filters:quality(80)/cloudfront-us-east-2.images.arcpublishing.com/reuters/CVNYPY5GUFPSXFBBXXBK6YLC4U.jpg",
        "publication": "Reuters",
        "id": 780,
        "date": "2022-03-29T09:06:00Z"
    },
    {
        "title": "Cryptoverse: Buoyant bitcoin helps market cruise past $2 trillion - Reuters",
        "imageurl": "https://www.reuters.com/resizer/hQUFyg89hsElzlhGfkPIvaoHxbY=/
```


11.1 Request

```
GET /api/twitter/search?coin=Bitcoin&username=elonmusk HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [USER_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    [
        {
            "id": 1503222294277197829,
            "id_str": "1503222294277197829",
            "conversation_id": "1503123611988766730",
            "datetime": "2022-03-14 04:11:38 GMT",
            "datestamp": "2022-03-14",
            "timestamp": "04:11:38",
            "user_id": 44196397,
            "user_id_str": "44196397",
            "username": "elonmusk",
            "name": "Elon Musk",
            "place": "",
```


11.2 Request

```
GET /api/twitter/search?coin=Bitcoin&username=elonmusk HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
```

Response

Body Headers (5) Status Code 401 Unauthorized

Pretty Raw Preview JSON


```
{
    "detail": "Failed to validate credentials"
}
```


12.1 Request

```
GET /api/users/ HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    {
        "id": 1,
        "first_name": "Arthur",
        "last_name": "Robertson",
        "email": "arthurrobertson2004@gmail.com",
        "hashed_password": "$argon2id$v=19$m=102400,t=2,
            p=8$cq5VSundG4MwRggh5FzrvQ$8fEX2CM+FDI5FUWquNbY2A",
        "admin": false
    },
    {
        "id": 2,
        "first_name": "test",
        "last_name": "test",
```


12.2 Request

```
POST /api/users/ HTTP/1.1
Host: {{baseUrl}}
Content-Type: application/json
Accept: application/json
Authorization: Bearer [ADMIN_TOKEN]
Content-Length: 158

{
  "email": "testingemail2@aqa.org.uk",
  "password": "SecurePassword123!",
  "first_name": "testing",
  "last_name": "testing example",
  "admin": "false"
}
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 {
2     "email": "testingemail2@aqa.org.uk",
3     "admin": false,
4     "first_name": "testing",
5     "last_name": "testing example",
6     "password": "SecurePassword123!"
7 }
```

13.1 Request

```
1 GET /api/users/10 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
4 Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1 {
2     "id": 10,
3     "first_name": "Demo",
4     "last_name": "Account",
5     "email": "demo@arthurr.co.uk",
6     "hashed_password": "$argon2id$v=19$m=102400,t=2,
          p=8$FIIwRihFiFFKifFeC0EIYQ$TJWSXi2xmyz2aqgWk2qJGw",
7     "admin": false
8 }
```

13.2 Request

```
1 PUT /api/users/10 HTTP/1.1
2 Host: {{baseUrl}}
3 Content-Type: application/json
4 Accept: application/json
5 Authorization: Bearer [ADMIN_TOKEN]
6 Content-Length: 178
7
```

```
8  {
9    "password": "testing",
10   "email": "newemail@email.com",
11   "new_password": "newpassword",
12   "admin": false,
13   "first_name": "labore aute dolor",
14   "last_name": "sunt labore"
15 }
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1  {
2      "password": "testing",
3      "email": "newemail@email.com",
4      "new_password": "newpassword",
5      "admin": false,
6      "first_name": "labore aute dolor",
7      "last_name": "sunt labore"
8  }
```

13.3 Request

```
1 DELETE /api/users/10 HTTP/1.1
2 Host: {{baseUrl}}
3 Accept: application/json
4 Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON

```
1  10
```

14.1 Request

```
1 GET /api/users/94/profile HTTP/1.1
```

```
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [USER_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
{
    "id": 94,
    "first_name": "Admin",
    "last_name": "Account",
    "comments": [
        {
            "id": 33,
            "news_id": 814,
            "content": "testing comment",
            "date": "2022-04-06 16:06:55",
            "title": "test"
        }
    ]
}
```


15.1 Request

```
GET /api/users/admins HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    {
        "id": 6,
        "first_name": "Admin",
        "last_name": "Account",
        "email": "admin@admin.com",
        "hashed_password": "$argon2id$v=19$m=102400,t=2,
            p=8$nFPKuVdqLeU859zb29s7Bw$dk27NUsUNa9RrqiJOuyAlA",
        "admin": true
    },
    {
        "id": 52,
        "first_name": "Arthur",
        "last_name": "Robertson",
```


15.2 Request

```
GET /api/users/count HTTP/1.1
Host: {{baseUrl}}
Accept: application/json
Authorization: Bearer [ADMIN_TOKEN]
```

Response

Body Headers (4) Status Code 200 OK

Pretty Raw Preview JSON


```
[
    {
        "count": 62
    }
]
```


XSS and SQL Injection Testing

I have tested a selection of routes explicitly for XSS and SQL injection, in addition to further tests.

Logging In I tested the login endpoint extensively with several SQL injection strings. All of them failed to work.

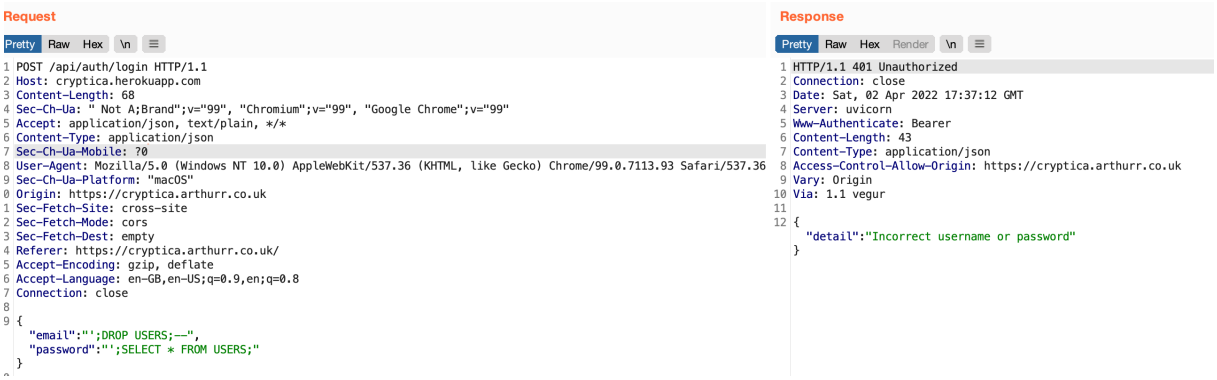

Figure 50: Screenshot of SQL Injection test

Registering I attempted to register with an account that included a common SQL and XSS injection string. I was able to create the account, however the SQL and XSS was not executed. As you can see below, the `<script>alert()</script>` is not embedded into the page, meaning that the application has successfully protected against injection.

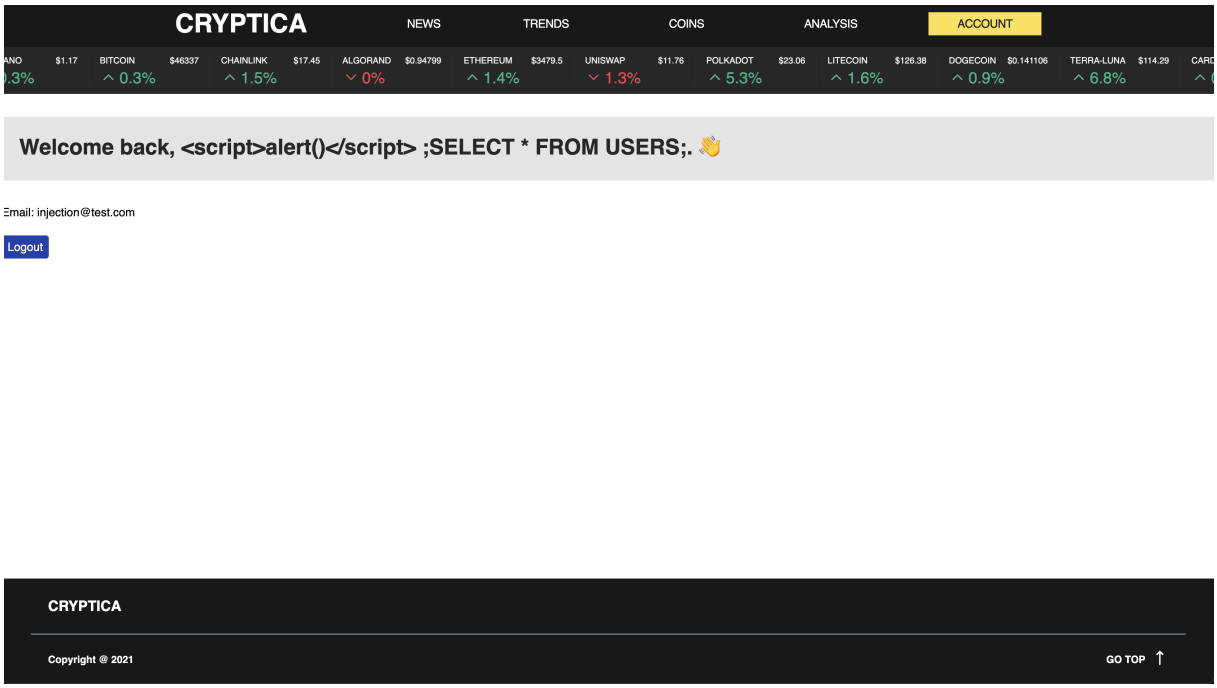

Figure 51: Screenshot of XSS Injection test

Posting Comments I tried inputting a variety of injection inputs to attempt to inject SQL or JavaScript into my page. All of them failed, and the SQL and XSS injection protection proved successful.

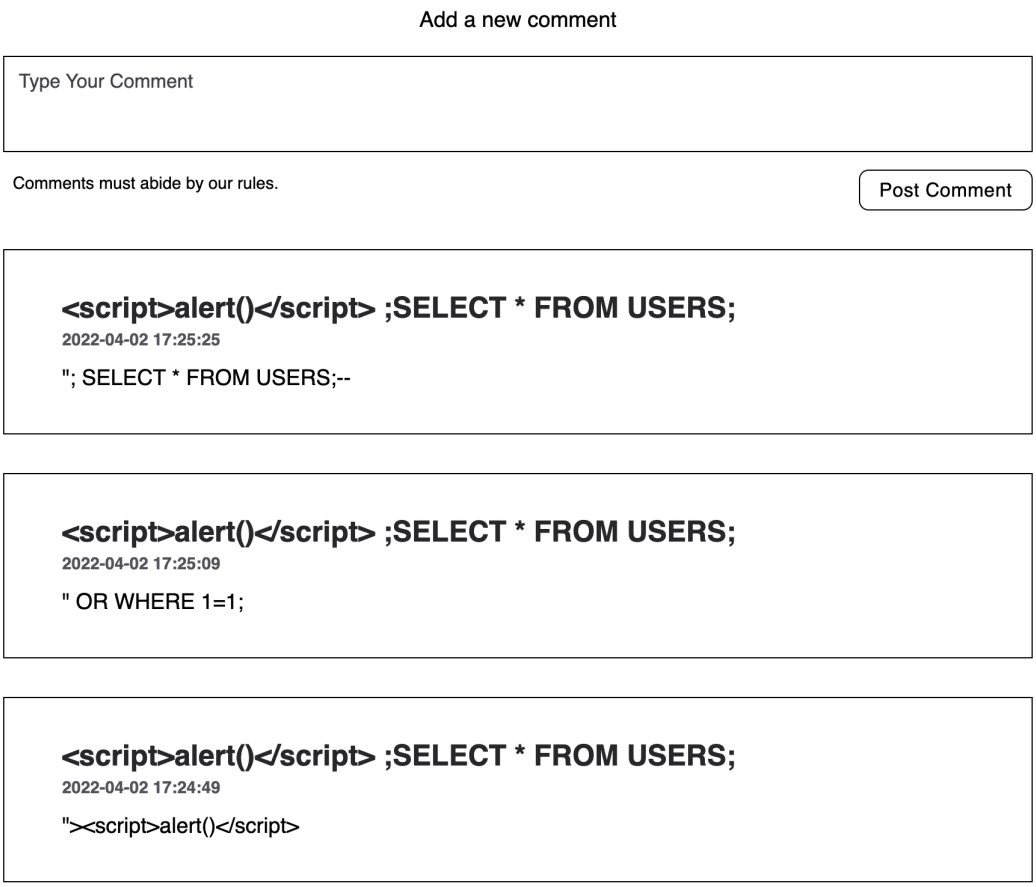

Figure 52: Screenshot of XSS Injection test

EVALUATION

OBJECTIVE COMPLETION

When I worked through the implementation and design phase of my project, I kept my objectives in mind at all time. This allowed me to complete almost all of them to a successful level.

You can see the table below where I have

Objective	Status
There should be a publicly accessible web application that allows the client to access the etc	Success, website is publicly accessible.
There should be two main components to the web application, a front end client and a back end API. The front end client should interact with the API and should be what the user interacts with. The API should handle all the fetching/processing of data, as well as any other functionality such as database management with CRUD.	Success, there is a separate API server and Client application.
The API should be capable of securely authentication users for the front end app. My client has specified that he would like anyone to be able to login and register for an account, so that he can share the application with his like-minded friends. Therefore, the API should be capable of handling multiple concurrent users having accounts, and should be able to authenticate and distinguish between them.	Success, there is login and registration functionality implemented. Support for multiple accounts.
The API should interact with the front end client application to ensure that users remain authenticated between sessions. Users should be able to login and then have to not enter their password again for a reasonable amount of time. This could be done through a method such as sessions, or cookie token generation.	Success, the client application makes requests to the API automatically. The API generates a JSON Web Token for authentication which is stored in the user's cookies.
The RSA algorithm should be used to sign the JSON Web Tokens. To do this, I will need to have an RSA key. Part of the program should be able to create RSA keys for use in this functionality.	Success, a valid and secure RSA Key is generated and used in the program.
The users data and passwords should securely be stored in a database. A secure, modern password hashing algorithm should be used, that uses hash salting to protect against attacks.	Success, the Argon2 password hashing algorithm has been used, which uses salting.

Objective	Status
The API should be capable of fetching and processing a specified users tweets from Twitter's API. It should be able to perform sentiment analysis on the tweet's content, and return the information to the user.	Success, the API is capable of fetching tweets and the client successfully displays them.
The API should have a database table that stores a collection of recent relevant news articles. The frontend client should then be able to display these articles for easy access. The news articles should ideally come from a variety of sources through web scraping. Only a brief excerpt of the article needs to be stored and displayed - to read the full article the users should be directed to the original site. Alternatively, the news articles should be fetched from an existing third party API that offers a service.	Partial Success, there is a database table containing hundreds of fetched news stories from an API. Building a website scraper for multiple different news sites was not feasible. The articles are successfully displayed to the user.
There should be a page that displays a list of the top 50 coins by market cap. It should display live data showing the price and other statistics about the coins. You should be able to click on any of the coins and it should take you to another page, showing further information about the coins performance. This should include a graph of the coins performance over time, and a brief description of the coin. In addition, on the specific coin page it should show a list of relevant articles stored in the database relating to the coin. If no such articles are found, it should not display any.	Success, there is a coins page that displays live cryptocurrency price data. Users can click and visit a page about a specific coin, and see related news articles.
Logged in users should be able to comment on any of the news articles, and anyone should be able to view said comments. Admin accounts should be able to delete any users comments, and users should be able to delete their own comments.	Success, the API is capable of checking if a user is authenticated. Only logged in users can comment. In addition, users can delete their own comments, and admin accounts can delete any users comments.

Objective	Status
There should be a basic profile functionality. Users should be able to view a users profile, and view information such as all their historic comments on articles.	Success, users can click on each others names to view a profile containing all a users comments.
The application should be secure against malicious parties. It should not be vulnerable to common flaws such as SQL or XSS (cross site scripting) injection, and users should not be able to bypass authentication methods implemented, e.g. viewing pages that are behind an authentication wall.	Success, in the testing phase I performed checks for SQL and XSS injection against the API. Users are also not able to bypass the authentication wall.
The application should contain analysis page for users tweets, that allows someone to input a users Twitter username. Then, they should be able to view a list of tweets, and should be able to see information on how the tweet has impacted the cryptocurrency market. It should display a candlestick graph that displays the price of the relevant cryptocurrency before and after the tweet. This page should also show the predicted sentiment of the tweet - whether the tweets content is positive or negative.	Success, the tweet analysis page successfully displays the information required.
The analysis page should also offer some basic analysis on the user's Twitter account as a whole. It should be able to produce a heat map of the time the user is typically active on Twitter, based on the time the user has tweeted previously. It should also display what device the user uses in the form of a pie chart, for example if the user is tweeting from an iPhone or from a computer.	Success, the user analysis page successfully displays the information required.

Objective	Status
The tweets analysis page should be able to perform some basic sentiment analysis on the user's tweets contents. It should attempt to estimate whether a tweet is positive or negative, and this should then be displayed to the user. For this, a neural network should be implemented using a Python deep learning library such as TensorFlow. The neural network should aim to have an accuracy of around 75%+. This objective is ambitious and primarily an extension that I should complete if I have enough time. Failing that, it should use an existing third party API for analysing sentiment, rather than creating my own sentiment model.	Success, the API returns a value for sentiment based on the response from the model. The model has an accuracy of approximately 85%, as evaluated by the test function.
The website should be fast to respond. This can be measured using Google's Lighthouse page score metrics, which is a service that returns a value on how fast the page performs. I want to aim for a score of 90-100, which is considered 'excellent'.	Success, the page score is above 90 as seen in the testing phase.

FEEDBACK FROM CLIENT

The feedback from my client was all positive. They expressed their appreciation of all their requests being implemented, and asked for no additional changes at this time.

POSSIBLE EXTENSIONS

The application has capabilities to be developed further if required by my client. There are several different extensions that could be implemented:

- As mentioned in the design phase, there are several additional security measures that could be implemented. One that I would recommend to be completed as a priority is some form of multi-factor authentication to offer an additional layer of protection to the applications users.
  - This would be complicated to implement, as would involve partially recreating the authentication flow. Measures such as email and SMS authentication would

likely require the use of a third party API, which would likely cost money. Whilst the complexity and difficulty of this extension is high, it is of quite high importance so should ideally be implemented as soon as feasible.

- The social aspect of the application could be developed further. Additional functionality such as the ability for users to post on a forum could be added.
  - This would simply involve creating a few more API routes that take advantage of the database class. This should not take too long to implement.
- The program could be expanded to be able to analyse stocks and other investments outside of cryptocurrencies.
  - The main challenge prevention an expansion to other commodities such as stocks, is the lack of freely available price data. If there was a budget and a subscription to a paid stock data API was purchases, this would be fairly easy to implement. However, it would be difficult to implement without a cost.
- The administration utilities could be expanded. Additionally functionality such as the ability to ban and delete users accounts directly from the application could be added.
  - This would require the creation of several new CRUD routes, which should be fairly easy to implement by making use of the existing classes.
- Further logging and analytics could be implemented. The application could integrate with Google Analytics or any other analytic tool to provide information about the demographic and quantity of users using the application.
  - This would be fairly easy to implement - to add Google Analytics to the program it would just require the provided analytics JavaScript tracking code to be injected to the head section of each page.
- A mobile application could be developed for iPhone and Android. The backend API functionality would not need to be changed for this, it would simply require the development of a mobile client.
  - This would be fairly easy to implement. There are many frameworks now such as React Native, that allow the easy conversion of websites to mobile applications.
- The random number generator in the RSA key generation could be upgraded to use a CSPRNG (Cryptographically-secure Pseudorandom number generator)

- This would be complex to implement, but should be considered very important for any program of enterprise level security. The default random number generator is not “true random”, which in some rare situations make it vulnerable to attacks.