



School of Electrical Engineering and
Telecommunications

SENG3011
Advanced Software Engineering
Workshop

Term 1, 2023
Event Intelligence Application

Final Design Report

Kyle Wu (z5363490)
Nathan Chow (z5255434)
Elvin Lee (z5309679)
Wenjie Liu (z5186632)
Swetha Ravichandran (z5365331)
Ethan Haffenden (z5367576)

Report Specification and Marking Criteria

Criteria	Weighting	Description/Definition of Done
Introduction	0%	<ul style="list-style-type: none"> Provides an overview of the contents of the report. Provides a brief overview of the service designed by the team. Outlines key objectives the service produced addresses.
Market Research/Business Value	15%	<ul style="list-style-type: none"> Discusses the background behind the targeted market. Identifies a gap within the targeted market. Outlines the solution used to address the gap identified. Identifies what part of the ecosystem the service will provide value to. Explains the value the service provides. Does the proposal make a compelling business case?
Requirements (Backend)	15%	<ul style="list-style-type: none"> Outlines the functional and non-functional requirements of the service's backend functionality. Includes user stories where relevant corresponding to features which will provide business value. Includes acceptance criteria where relevant defining when a feature has successfully been completed.
Requirements (Frontend)	15%	<ul style="list-style-type: none"> Outlines the functional and non-functional requirements of the service's frontend functionality. Includes user stories where relevant corresponding to features which will provide business value. Includes acceptance criteria where relevant defining when a feature has successfully been completed.
Software Architecture	20%	<ul style="list-style-type: none"> Provides an analysis of the service's high-level architecture. Justifies all technologies and languages used in the construction of the service. Provides a software architecture diagram as a visual aid explaining the services architecture. Provides an explanation of the services design. Provides at least 2 sequence diagrams to model the end-to-end flow of the service.

Maintenance strategy	5%	<ul style="list-style-type: none"> • Provides an overview of the deployment strategy. • Provides an overview of the observability strategy.
Testing	10%	<ul style="list-style-type: none"> • Outlines the testing performed on the team's service. • Explains the team's approach to testing another team's service. • Provides an explanation for how tests have been integrated into another team's service.
Integration of another service	5%	<ul style="list-style-type: none"> • Justifies the integration of another team's service. • Provides evidence of using another team's service.
Challenges and constraints	10%	<ul style="list-style-type: none"> • Provides an overview of the technical and non-technical challenges the team has faced. • Identifies constraints technical and non-technical constraints of the project. • Provides evidence of risk management and strategies.
Conclusion	0%	<ul style="list-style-type: none"> • Summarises the findings of the report.

Table of Contents

1. Introduction.....	7
2. Market Research & Business Value	8
2.1 Overview.....	8
2.2 Background.....	8
2.2 Solution.....	8
2.3 Business Value.....	8
2.3.1 General Business Values.....	8
2.3.2 Value to the Ecosystem.....	9
3. Backend Requirements	10
3.1 Overview.....	10
3.2 Non-Functional Requirements	10
3.2.1 Core Non-Functional Requirements	10
3.2.2 Vitality Non-Functional Requirements.....	11
3.3 Functional Requirements	12
3.3.1 Fetch API	12
3.3.2 Upload API	13
3.3.3 Export API	14
4. Frontend Requirements.....	15
4.1 Overview.....	15
4.2 Non-Functional Requirements	15
4.3 Functional Requirements	16
4.3.1 Register Page	16
4.3.2 Login Page	16
4.3.3 Overview Page.....	16
4.3.4 Company Search Page	17
4.3.5 News Search Page.....	18
4.3.6 Interactive Chart Page.....	19
4.3.7 Miscellaneous	20
5. Software Architecture	21
5.1 High-level Software Architecture	21
5.1.1 Requirements	21
5.1.2 Infrastructure Analysis.....	21
5.1.3 Conclusion	22

5.2 Tech Stack.....	23
5.2.1 Service Layer	23
5.2.2 Database Layer.....	23
5.2.3 API Layer.....	23
5.2.4 Frontend Layer.....	24
5.2.5 Observability Layer	24
5.2.6 Feature Flags.....	24
5.2.7 Deployment Layer (Backend).....	24
5.2.8 Deployment Layer (Frontend)	24
5.3 Software Architecture Diagram	25
5.4 Service Design	25
5.4.1 API Interface.....	26
5.4.2 Data Modelling	26
5.4.3 Data Requirements.....	26
5.4.4 Data Model.....	27
5.4.5 Data Storage.....	28
5.5 System Modelling	29
5.5.1 User logging into the frontend application	29
5.5.2 User is logged in and is viewing financial news data for a specific company.....	30
5.6 Web Application Design.....	31
5.6.1 UI/UX Considerations	31
5.6.2 Authentication.....	34
5.6.3 Communication with Backend.....	35
5.6.4 Frontend Testing	36
5.6.5 Deployment.....	37
5.7 Future Design Plans	39
5.7.1 Caching	39
5.7.2 Price Prediction with Machine Learning	39
6. Maintenance Strategy.....	40
6.1 Observability Strategy	40
6.1.1 Observability data types.....	40
6.1.2 Three-phase strategy	40
6.1.3 Observability Implementation	41
6.2 Deployment Strategy	47

6.2.1 Feature Flag Implementation	47
6.2.2 Our Flag	48
7. Testing.....	48
7.1 Testing our Microservice	49
7.2 Testing H09A_FOXTROT's microservice.....	51
7.2.1 What and why did we test it.....	51
7.2.2 Main components of what we tested.....	52
7.2.3 How tests were incorporated.....	52
8. Ecosystem Service Integration.....	54
8.1 Overview	54
8.2 Justification	54
8.3 Evidence of Use	54
9. Challenges and Constraints	56
9.1 Overview	56
9.2 Technical constraints	56
9.2.1 Third-party Compatibility and Dependencies.....	56
9.2.2 Knowledge and Inexperience.....	56
9.3 Non-Technical Constraints	57
9.3.1 Data Quality and Integrity	57
9.3.2 Data Scarcity and Abundance.....	58
9.3.3 Time Limitations.....	58
9.3.4 Scope Creep	59
9.4 Risk Management	59
10. Conclusion	62
11. References.....	63
12. Appendix.....	63

1. Introduction

In this report, we will provide an in-depth explanation and justification of the design and implementation of our web application, FinConnect. This will include sections discussing the process of market research and the ideation of the service, the key business values behind the application, the requirements drawn from these business values and the decisions behind the implementation of FinConnect, including considerations made to minimise constraints and risks. Through this report, we seek to offer insights into the design processes behind the construction of the application and its integration into the company ecosystem.

Initially, FinConnect is a comprehensive financial news data platform designed to provide users with up-to-date and accurate information on the latest trends of companies within the financial market. FinConnect was developed to offer a user-friendly and intuitive interface, that caters to the needs of novice investors, allowing them to better understand the impacts of global events on the financial market.

In line with this goal, the following key objectives were identified:

- Provide users with real-time and easy-to-digest financial news data from reputable sources.
- Provide users with valuable insights and analysis.
- Ensure a user-friendly and intuitive interface.
- Integrate seamlessly into the company ecosystem.

2. Market Research & Business Value

2.1 Overview

Market research is an essential steppingstone when it comes to understanding the competitive landscape, identifying customer needs, and developing a successful service. In this section, we will discuss the background behind our application, along with our proposed solution and its respective business values.

2.2 Background

The financial technology and analytics market has been experiencing significant growth in recent years, as evident by the projected annual growth rate of 14.7% from 2020 to 2027 (Grand View Research, 2022). This growth can be attributed to advancements in technology, particularly in the areas of artificial intelligence and machine learning, which have allowed financial institutions and investors to leverage data-driven insights for making informed decisions and gaining a competitive edge (OECD, 2021). However, traditional systems such as Bloomberg, which have long been considered the industry standard, pose challenges for non-professional investors, due to their high costs and complexity, creating a barrier to entry and leaving a noticeable gap in the market.

2.2 Solution

To fill the identified gap in the financial technology and analytics market, we have developed FinConnect as a low-barrier-to-entry, easy-to-use solution, that allows users to access real-time interactive financial news charts, which helps them identify patterns and fluctuations in stock prices and correlate them to relevant events. To support the operations of the front-end application, we have also created three separate financial APIs: Fetch, Upload, and Export. These APIs provide value by offering seamless access to financial data associated with companies, allowing users to fetch, upload, and query data stored in the S3 data lake, using GraphQL. The APIs are designed to enhance the accuracy and speed of data retrieval and analysis, addressing the needs of investors who require up-to-date and reliable information for making informed investment decisions.

2.3 Business Value

2.3.1 General Business Values

The FinConnect financial news platform and its associated APIs provide several business values to the average user. Firstly, the application offers users comprehensive and reliable financial data, including up-to-date stock prices, news articles and market trend indicators, allowing them to understand the various aspects of the market and how they impact each other. Secondly, the service improves the speed and accuracy of financial news data retrieval, providing users with real-time access to relevant data and automatically filtering articles relevant to the user's needs. Finally, by offering a more accessible and affordable solution for non-professional investors, the application

helps to democratize the financial market, by removing unfair barriers to entry and allowing them to actively participate in the market with confidence.

2.3.2 Value to the Ecosystem

In addition to the general business values, the solution proposed offers additional specific values to the company ecosystem, by providing users within the company with direct access to the aggregated data and additional endpoints. Through scraping and correctly formatting the aggregated data, the solution provides users in the ecosystem with the ability to perform their own complex analyses, as all data matches the predefined event dataset structure. This ensures that users within the company no longer need to rely on external sources, which will save them time and effort during their data collection. Lastly, all backend APIs, allow for seamless integration with existing company systems, providing a streamlined and efficient workflow for data retrieval and utilisation.

3. Backend Requirements

3.1 Overview

Developing the backend for FinConnect required careful consideration of its functional and non-functional requirements. It is essential to understand these requirements as the backend is responsible for processing data, managing the application's business logic, and serving data to the front end and other third-party services. To ensure the backend provided business value, it was important to create user stories that correspond to features that adhere to these requirements. Furthermore, each user story must be accompanied by acceptance criteria to easily define when a feature has been completed. This also helps users of FinConnect understand what constitutes a completed feature. In this section, the non-functional requirements will be stated first, followed by functional requirements that will be described by the user stories and acceptance criteria that FinConnect's backend is defined by.

3.2 Non-Functional Requirements

3.2.1 Core Non-Functional Requirements

These are the core non-functional requirements of our platform microservice, which encapsulates most of our backend service.

Performance Efficiency

The performance efficiency of the APIs is largely decided by the efficiency of the S3 bucket, as in this sprint the main functionalities of the APIs are exporting and uploading to/from the S3 bucket. The S3 bucket uploads at 2.3 Mbps, so all JSON uploads should take less than a second to execute. The capacity also depends on the S3 bucket and should be able to store as many JSONs as the S3 bucket allows.

Security

Authentication and Authorisation should be handled by the provided infrastructure, with the /sign_up and /login endpoints. A "group" key should also be provided with the "username" and "password" keys in the body. The "username" and "password" should be confidential, but the "group" key should be retrievable and used when naming the JSON files that are uploaded to the S3 bucket.

Users should also be able to monitor the activity of the service through status codes and checking the activity of the S3 bucket. Logs on Amazon CloudWatch should also be available for more in-depth monitoring.

Reliability and Safety

As long as the API is deployed, it should be accessible at all times. However, in cases of faults and errors, they should be handled with status codes and should be logged on CloudWatch for further investigation.

External libraries and packages should also be managed and made sure they pose no risk. The following external libraries were Graphene, which should be used for building GraphQL APIs, boto3, which should be used to upload to the S3 bucket, and requests which should be used for testing and using endpoints. All should be deemed safe to use and should be updated to their latest versions.

Maintainability

The export and upload are fundamental APIs that should be developed with expansion in future sprints in mind. These APIs should not be changed after Sprint 1 however and should be closed from any additional modifications.

The APIs should be a platform for other groups to export and upload JSON data from/to the S3 bucket. This should be enforced by adding an entry to the confluence regarding the functionality of our API, as well as creating a discord for future users to join in case of questions and discussions regarding how to use it.

The APIs should also be tested with HTTP requests and manually through checking status codes with Postman and the S3 web console for successful uploads.

Portability

Though the API focuses on exporting and uploading financial data, if some other JSON data follows the specified JSON format of the ecosystem, then it should also be able to be uploaded to the S3 bucket using our APIs, thus allowing portability to other microservices that may not use financial data.

The repository should also be able to be viewed through the “cseteaching-unsw-edu-au” organization on GitHub and can be cloned by users.

3.2.2 Vitality Non-Functional Requirements

These are the non-functional requirements for Sprint 3’s Vitality Initiative, where our backend service was updated to implement caching.

Performance

The cache retrieves data at least 25% faster than direct access to the S3 bucket. There would not be any point in incorporating it if it did not perform better than what we have already implemented.

Furthermore, a key objective of our service is to provide high-speed access to financial data. Implementing caching will help achieve our goal about highly demanded data.

Reliability

The cache operates without failure at least 90% of the time over 1 month. Ensuring that the service is reliable is extremely important to our end users. Incorporating the caching system should not fundamentally break the way our service operates, hence by incorporating caching, the service should still be highly performant. We have allowed a minimal margin of error when the cache fails as a result of the team's experience with the technology. The use of feature flagging, logging, New Relic Metrics, and New Relic Alerts will assist with ensuring that the service's reliability is highly reliable.

Scalability

The cache must be able to manage and store the historical data from up till the 1st of January 2010 for at least 1 company, with horizontal or vertical scalability options to allow for up to 25 companies to be cached. As the service continues to grow, it is vital that it is scalable to ensure that additional resources can be quickly provisioned to handle any increase in usage. For the cache, this is particularly important as we plan on caching the most frequently requested data. Currently, we have been testing small sets of data, however as the service expands, demand for specific datasets will inevitably increase, which would benefit users significantly if they were cached. Ensuring that both horizontal and vertical scaling options are critical to maintaining fast data access speeds in the future, which will be determined by the caching service provider. As a result, the decision of which caching service to use will be heavily informed by the scalability requirement.

3.3 Functional Requirements

3.3.1 Fetch API

User Story 1:

As a financial analyst, I want to be able to retrieve historical stock price data from a specific company and a given time range, so that I can analyse the company's performance in the past.

Scenario: Retrieving historical stock price data.

Given: The user has access to the API.

And: The user has entered the company ticker into the query.

And: The user provides a start date in the query.

When: The user sends the query to the Fetch API.

Then: The API returns a JSON object containing historical stock price data.

And: The JSON object should be correctly formatted.

And: The JSON object should contain a new time attribute containing the timestamp and location of the object's creation.

3.3.2 Upload API

User Story 2:

As a financial analyst, I want to be able to store my corrected JSON object in the S3 data lake, so that I can retrieve it through the EXPORT API.

Scenario: Uploading a JSON object into the central database.

Given: The user has access to the API.

And: The user has entered a valid JSON object into the query.

When: The user sends a request to the Upload API.

Then: The API should store the JSON object inside its database.

And: The API should return a confirmation message, showing the storage was a success.

User Story 3:

As a financial analyst, I want to be able to store my corrected JSON file in the S3 data lake, so that I can combine multiple related files inside the system.

Scenario: Uploading a JSON file into the central database.

Given: The user has access to the API.

And: The user has entered a valid JSON file into the query.

When: The user sends a request to the Upload API.

Then: The API should store the JSON file inside its database as a JSON object.

And: The API should combine the object with other related objects.

And: The API should return a confirmation message, showing the storage was a success.

3.3.3 Export API

User Story 4:

As a financial analyst, I want to be able to retrieve historical stock price data from a specific company and a given time range, so that I can quickly access and analyse the company's performance.

Scenario: Retrieve a whole JSON file from the S3 data lake.

Given: The user has access to the API.

And: The user has entered the company ticker, start date, and end date into the query.

When: The user sends a request to the Export API.

Then: The API returns a JSON object containing historical stock price data.

And: The JSON object should be correctly formatted.

User Story 5:

As a financial analyst, I want to be able to retrieve specific historical stock price attributes given a specific company and a given time range, so that I can query specific data attributes I require.

Scenario: Retrieve a partial JSON file from the S3 data lake.

Given: The user has access to the API.

And: The user has entered the company ticker, start date, and end date into the query.

And: The user has entered a list of attribute names.

When: The user sends a request to the Export API.

Then: The API returns a JSON object containing the specific attributes of the queried historical stock price data.

And: The JSON object should be correctly formatted.

4. Frontend Requirements

4.1 Overview

To correctly construct a user interface and offer an optimal user experience, it is essential to understand the details of the non-function and functional front-end requirements. In this section, we will begin by covering the various non-functional requirements, followed by the functional requirements which will be shown through the user stories and acceptance criteria that FinConnect's front-end application was defined by.

4.2 Non-Functional Requirements

Usability

Usability is critical to creating systems that are designed for human use and is essential to displaying information in a way that avoids frustration or confusion from end users. The front-end application should be simple and easy to use. The entire front end should consist of no more than two pages to avoid complicating the functionality of our service. Usability should be described with user stories to better understand the use of the system and how quickly they can learn it.

Performance

With our services fetch and upload alongside FOXTROT's fetch from the Guardian and New York Times APIs, the entire process of applying filters and displaying the graph is expected to fluctuate in time. However, regardless of the size of the dataset and the filters provided, it should resolve in acceptable response times of around 2 seconds. The front end should also be concurrent, allowing multiple users to access the application without any detrimental effects on its performance. The application should also consider how downtime or issues are handled to ensure that users can access the system as soon as possible. Every time the front-end calls the API from the back-end, to get a good performance, it is used by the AJAX to load the part of the data at first.

Compatibility

The front end should also have compatibility in mind regarding usability and performance. The front end should be usable in a multitude of devices and accommodate all types of end users. This can include implementing dynamic screen sizes, text-to-speech, and language filters that only return articles in a specific language. Compatibility with FOXTROT's service should also be prioritized, to ensure that there are no bugs and that data fetched from both services can be used in conjunction with each other and provide meaningful information.

Scalability

The front end should be designed with scalability on both performance and usability in mind. Performance should be scalable by considering workload increases and the availability of the

service. The usability of the service should also be scalable by allowing for expansion when new potential functionalities are introduced. This can be done by making use of the adaptability that ReactJS provides, as well as ensuring to use of any packages or templates that also have scalability in mind.

4.3 Functional Requirements

4.3.1 Register Page

User Story 1:

As a user, I want to be able to register for an account so that I can access the contents of the application and ensure that my user data is secure.

Scenario: Registering an account on FinConnect.

Given: The user has entered their email, password and confirmed password into the register form.

When: The user clicks the “Register” button.

Then: The system will redirect the user to the “Overview” page.

4.3.2 Login Page

User Story 2:

As a user, I want to be able to log in to my account so that I can access my user data and the services of FinConnect.

Scenario: Logging into an account on FinConnect.

Given: The user has registered an account.

And: The user has entered the email and password into the login form.

When: The user clicks the “Login” button.

Then: The system will redirect the user to the “Overview” page.

4.3.3 Overview Page

User Story 3:

As a new user, I want to view all of the features available to me, so that I can understand what the application does.

Scenario: Viewing the features of the application.

Given: The user is logged out.

And: The user has already signed up.

When: The user enters their details

And: The user clicks on the “Sign In” button.

Then: The system should display the “Overview” page, containing the features of the application.

And: The system displays the list in a user-friendly and visually appealing format.

4.3.4 Company Search Page

User Story 4:

As a new user, I want to be able to access the OHLC data of the top 10 companies, so that I can view the general state of the market.

Scenario: Accessing the OHLC data of the top 10 companies by market capitalisation.

Given: The user is logged in.

And: The user is on the “Overview” page.

When: The user selects the "Company Search" button in the sidebar.

Then: The system should display a list of the top 10 companies by market capitalisation.

And: The list should include their ticker and OHLC data for the current day.

User Story 5:

As a financial analyst, I want to be able to search for the OHLC data of any company, so that I can see if they are a good investment.

Scenario: Accessing the OHLC data of a specific company, given a company ticker.

Given: The user is logged in.

And: The user is on the “Company Search” page.

When: The user inputs a company ticker into the search bar.

And: The user selects the “Search” button.

Then: The system should display a list with only the given company.

And: The list should include their ticker and OHLC data for the current day.

4.3.5 News Search Page

User Story 6:

As a financial analyst, I want to be able to have my search autocomplete, so that I can see all the possible companies I can search for.

Scenario: Autocompleting a search query for a company ticker.

Given: The user is logged in.

And: The user is on the “News Search” page

When: The user enters a partial ticker into the search bar

Then: The system returns a list of potential tickers, auto completing the query.

And: The list is given in order of the closest match.

User Story 7:

As a financial analyst, I want to be able to search for news articles associated with a particular stock or index, so that I can understand the fluctuations in price.

Scenario: Searching for news articles by market indexes.

Given: The user is logged in.

And: The user is on the “News Search” page

And: The user has entered a ticker into the search bar

When: The user clicks on the first autocomplete result in the list.

Then: The system returns a list of relevant news articles, containing their title, date of publishing and the link to the sources.

And: The system displays the list in a user-friendly and visually appealing format.

4.3.6 Interactive Chart Page

User Story 8:

As a financial analyst, I want to view the historical financial and news data associated with a company, so that I can identify patterns and trends over time and make more informed investment decisions.

Scenario: Viewing an interactive financial news chart.

Given: The user is logged in.

And: The user is on the “Company Search” page.

When: The user clicks the “Details” button for a given company.

Then: The system should display the financial and news data for that company inside of an interactive financial news chart.

And: The graph should be in the form of a candlestick chart.

And: The graph should include key metrics such as open, high, low, and close.

And: The table under the graph should include the news articles of the currently selected date.

User Story 9:

As a financial analyst, I want to filter and customise the interactive chart based on specific date ranges, so that I can focus on the impact of an event over a time period.

Scenario: Customising the range of interactive financial news chart

Given: The user has generated an interactive chart for a given company.

When: The user selects a range in the chart.

Then: The system should zoom in on the interactive chart to the given range.

And: The graph should be in the form of a candlestick chart.

And: The graph should include key metrics such as open, high, low, and close.

And: The table under the graph should include the news articles of the currently selected date.

4.3.7 Miscellaneous

User Story 10:

As a user, I want to be able to log out of my account, so that I can exit the FinConnect application safely.

Scenario: Logging out of an account on FinConnect.

Given: The user is logged in to an account on FinConnect

When: The user clicks the user avatar on the top right corner of the page.

Then: The system should display a dropdown menu below the user avatar.

When: The user clicks the “Sign Out” button

Then: The system will remove the token assigned to the user and unauthenticated their session.

And: The system will redirect the user to the “Login” page.

5. Software Architecture

5.1 High-level Software Architecture

The team has decided to build an import and export service, utilising a lambda architecture. From our research into financial data and financial markets, we have recognised that the service's target audience will be independent traders, professional traders, and large financial companies including, but not limited to investment banking firms, investment management companies, and market-making firms from across the globe. In the world of trading and finance, it is critical that access to financial data is fast and highly available to ensure that algorithms and analytics can be performed rapidly to inform investors and market makers before authorising critical, high-risk actions. Access to this data must also be globally available across all time zones due to the round-the-clock nature of financial markets, meaning that it is a strict requirement that the service must run at all times of the day. Furthermore, due to the sheer volume and size of financial data openly available to be consumed and analysed at any given time, considerations must be made about the cost of running the service and its ability to handle the volume of requests and size of data within each request. With these requirements in mind, from the team's analysis of different software architecture models, it has been decided that our service will be built following the lambda architecture for the following reasons.

5.1.1 Requirements

1. The service must facilitate acceptable data access speeds.
2. The service must run continuously regardless of location and time of day.
3. The service must be able to handle high volumes of requests and frequent updates.
4. The service must be able to handle and manage large volumes of data both to be stored and sent through in requests.
5. The service must be scalable and able to grow as the project does adapt to new environments and features as it is built out.

5.1.2 Infrastructure Analysis

Microservice analysis

Advantages	Disadvantages
Easily scalable services both horizontally as well as vertically	Increased risk of complications in the communications for microservice modules
Integration and development are streamlined and simplistic	Testing and debugging the microservice is complex and difficult to complete
Reduced concerns about the dependency on external sources	Implementing and deploying the service is complex and can prove challenging

Modular components within the microservice make it easy to understand	Requires the services to be running continuously putting resources to waste
---	---

Lambda analysis

Advantages	Disadvantages
Cost-efficient. Only pay for the time when the service is running.	Cold starts when the service is not used frequently. Not many developers can do to control this.
Focus more on the development of code. Managing deployment and scalability is not heavily required.	Remotely testing lambda functions are difficult, have to use Postman to work around
Fast speed of development and deployment time, are important for the short development cycle	Recreate lambda function if there is a failure point present
Modular design can easily be expanded upon, built up and understandable	There is a general payload limit on the services that are present in it
Lambda has high reliability if one function fails it doesn't break our system	Latency is of concern, as the current program is running at a 5-second distinction

5.1.3 Conclusion

The lambda architecture facilitates fast deployment and seamless, modular integration, as users can call the functions without the overhead of interacting with a microservice. As such, this ensures there is no single point of failure since all lambda functions can be redeployed without re-containerizing the entire system. Moreover, a critical pain point was the cost of deployment, which is addressed through using lambda functions, as the pay-as-you-go model fits our needs compared to a microservice, as it would require around-the-clock maintenance and hourly billing. For the following reasons we have identified that the lambda architecture will be a safer choice for the development of the import and export service which the team has decided upon.

However, in future it may be more optimal to deploy the service using a microservice architecture as lambdas may require a 'cold start', increasing the latency of accessing financial data which is critical for making high-risk financial decisions. Furthermore, microservices can process a greater amount of data requests, which will assist in the scalability and maintenance of the service.

5.2 Tech Stack

Note: This section of the report will provide a brief justification for the technologies being used. To find more details about the considerations the team has made when deciding on the tech stack, the following links have been provided.

- <https://unswcse.atlassian.net/wiki/spaces/F12AZULU/pages/112199299/Software+Architecture#Tech-Stack>
- <https://unswcse.atlassian.net/wiki/spaces/F12AZULU/pages/144429455/2.1+Engineering+Proposal#3.1.2-Technical-Stack>

5.2.1 Service Layer

The team has used Python to produce the backend for the application. Python was heavily favoured as all teams were familiar with developing web applications with Python, which would improve the speed of development for the service. Furthermore, Python has an abundance of libraries available to use for the development of our service. Pre-existing libraries existed that scrape data from our data source Yahoo Finance. Due to python's popularity, many other services which the team plans on using, have developed software development kits (SDK) for the Python language. The major drawback of using Python however are its slow performance, poor memory efficiency and dynamic typing which may result in unsafe code. When considering Python, the team discussed the effect of these drawbacks and concluded that they would not be of serious concern to the development of the service.

5.2.2 Database Layer

The team has decided to use AWS S3 for the database layer of the service. The course had already set up S3 buckets for the use which served as the central data lake for datasets to be uploaded into, and hence the team decided to simplify the handling of generated datasets by simply utilising the existing data lake. Furthermore, the S3 is highly scalable and capable of handling an extremely large amount of data which was fitting given the wealth and abundance of financial data which the service could potentially scrape.

5.2.3 API Layer

The API layer for the service will be GraphQL. GraphQL is an emerging technology, which provides advantages over more popular RESTful APIs. GraphQL allows for flexible queries under a single endpoint, allowing users to specify the specific data they require from a request reducing over and under-fetching of data which may reduce request latency. GraphQL is also strongly typed, which means that clients are guaranteed that data conforms to a specific format at compile time which will ensure the API is more reliable. GraphQL schema-focused design also makes modifying an API easier without having to make significant changes to the API causing it to break or function.

5.2.4 Frontend Layer

The team has chosen to use ReactJS and Tailwind CSS for the frontend layer. ReactJS was the primary frontend framework chosen by the team due to its simplified state management and enhanced developer experience. Tailwind CSS was chosen as a component library to build the web application due to its simplistic and minimal design, unified prebuilt components and rapid UI development. This would reduce the need for the team to build our components from scratch saving us time in the limited time frame we had to develop the web application.

5.2.5 Observability Layer

The team has used New Relic to implement observability into the service. New Relic has been provided as a service from the course and hence was the only consideration for this layer. It also has a Python SDK which the team can leverage to implement observability features. The service also provides a visualisation of the telemetry data collected and allows developers to set up alerts for when a service encounters a critical failure.

5.2.6 Feature Flags

The team has decided to use Flagbase to implement feature flags. This is a service provided by the course and hence was the only option considered for this layer. Flagbase recently produced a Python SDK which integrates well with the existing tech stack the team has already chosen. The service provides a simple interface which allows the team to roll out new or revert features without needing to redeploy the entire service.

5.2.7 Deployment Layer (Backend)

The team has decided to deploy with Amazon Lambda. This is mainly due to the requirements specified by the project specification. The main reason for choosing Lambda is due to the ease of deployment with the service. Furthermore, lambdas are highly available and scalable, which improves the team's ability to maintain the service as it continues to grow. Finally, lambdas are capable of handling a high volume of requests quickly, which by the nature of financial services, we have identified as a requirement of our service.

5.2.8 Deployment Layer (Frontend)

The team has decided to deploy the front end using an S3 bucket. This comes to ensure that the entirety of the service has been integrated within the same AWS ecosystem, all placed behind the provided AWS API gateway. S3 bucket allows for static website hosting with minimal cost and scalability. The service also has extreme reliability and availability thus guaranteeing the service's reliability.

5.3 Software Architecture Diagram

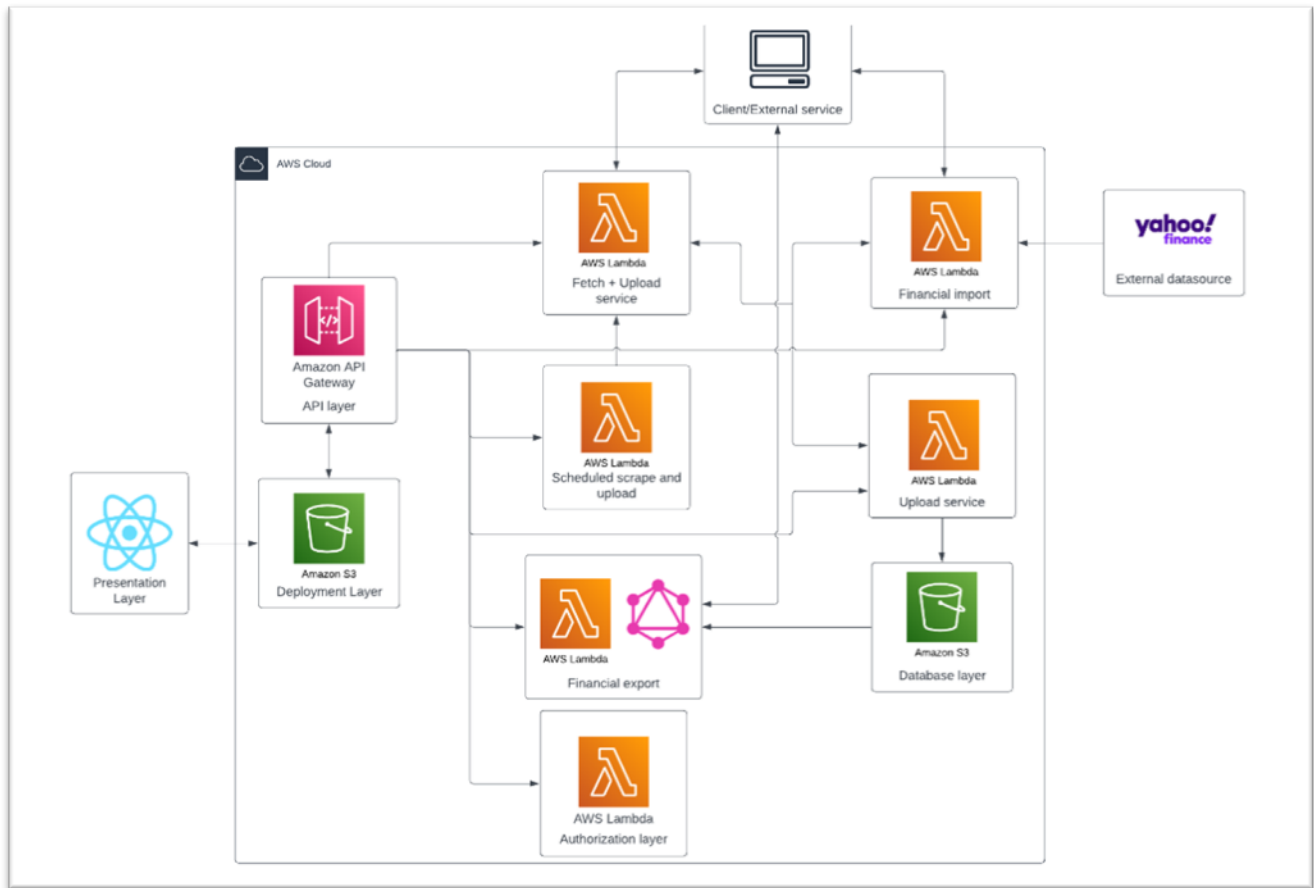


Figure 5.3: Software Architecture Diagram

5.4 Service Design

5.4.1 API Interface

To document how to communicate with the service, the team has created two forms of documentation explaining the service API interface. The first is the Swagger documentation which can be used to understand how to connect to the service via HTTP requests. The second is a website containing information on how to form GraphQL queries to retrieve data from our export service.

Swagger documentation:

https://app.swaggerhub.com/apis/F12A_Zulu/SENG3011_F12A_ZULU_Financial_database/1.0.0

GraphQL documentation:

https://seng-3011-f12-a-zulu-graphql-docs.vercel.app/f12a_zulu-financial-data-export-graphql-documentation

5.4.2 Data Modelling

As specified in the project specification, the team must collect data from an external source of our choice. The team has decided to collect financial stock data from the Yahoo Finance website. This section of the report identifies requirements that our data model must satisfy along with the data model the team has designed to fill these requirements.

5.4.3 Data Requirements

The project specifications also outline basic data conventions by which each dataset format should abide (linked here: [Data Model Specification](#)).

This format consists of the following three core structures:

1. A dataset identifier object consisting of the following attributes:
 - Data source identifying the source the data comes from.
 - Dataset type identifies the type of data stored in the dataset.
 - The dataset ID is a unique identifier for the dataset.
2. A timestamp object indicating the date and time of data collection consisting of the following attributes:
 - Timestamp identifying when the data was collected.
 - Time zone identifying the location the data was collected from.
 - Duration identifying the unit period in which data was collected.
 - Duration units identifying the units the duration was measured in.
3. An event object which outlines the data collection consisting of the following attributes:
 - Timestamp object identifying the date, time and duration the data was generated.
 - Event type identifying the type of data collected.

- The data attribute object consists of the attributes storing the data collected.

With the structure above in mind, the dataset collected should be able to be stored as a JSON file.

5.4.4 Data Model

As the dataset should be able to be stored as a JSON file, the team has decided to use a non-relational schema. The schema consists of three core data objects relating to the requirements outlined above. These data objects can be seen in **Figure 5.4.4**.

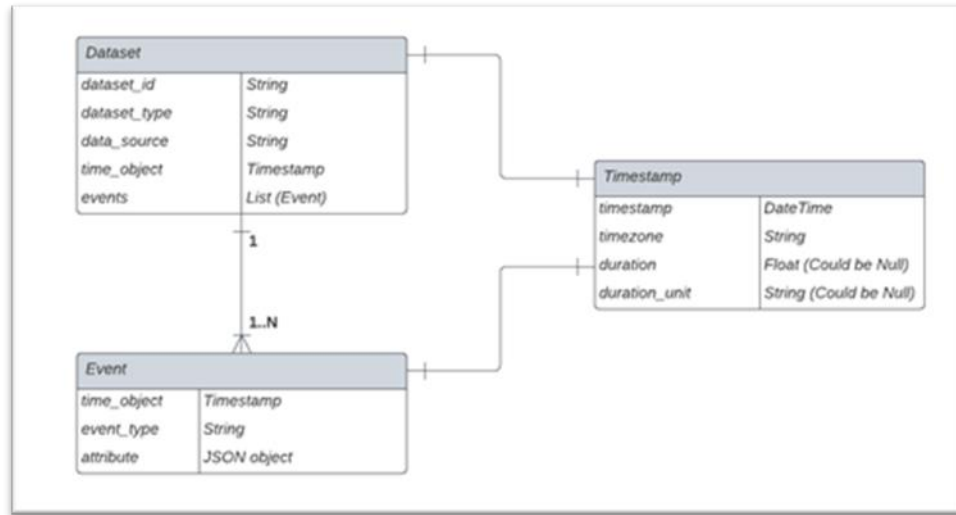


Figure 5.4.4: ER diagram of the Event Dataset Model

As an example, the final format for each JSON file should conform to the following example:

```

{
  "data_source": "yahoo_finance",
  "dataset_type": "Daily stock price",
  "dataset_id": "GOOGL",
  "time_object": {
    "timestamp": "2023-03-29 11:08:40.647562",
    "timezone": "GMT"
  },
  "events": [
    {
      "time_object": {
        "timestamp": "2010-01-04 14:30:00.000000",
        "timezone": "GMT",
        "duration": 1,
        "duration_unit": "day"
      },
      "event_type": "stock ohlc",
      "attribute": {
        "open": 15.6894388199,
        "high": 15.7535037994,
        "low": 15.6216220856,
        "close": 15.6844339371,

```

```
        "volume": 78169752,  
        "daily_change": 0.1694192886,  
        "daily_return": 0.0109196989,  
        "weekly_change": 0.2069568634,  
        "weekly_return": 0.0133714857,  
        "monthly_change": 1.0262756348,  
        "monthly_return": 0.0700139549,  
        "rsi": 79.463414876,  
        "ticker": "GOOGL",  
        "date": "2010-01-04 14:30:00.000000"  
    }  
}  
]  
}
```

5.4.5 Data Storage

To conform to the organisation's requirements for data storage in the data lake, the team will store datasets as JSON files and upload them to the organisation's S3 database. To do this the team has utilised the `Boto3` Python library to connect to the organisation's S3 bucket.

5.5 System Modelling

5.5.1 User logging into the frontend application

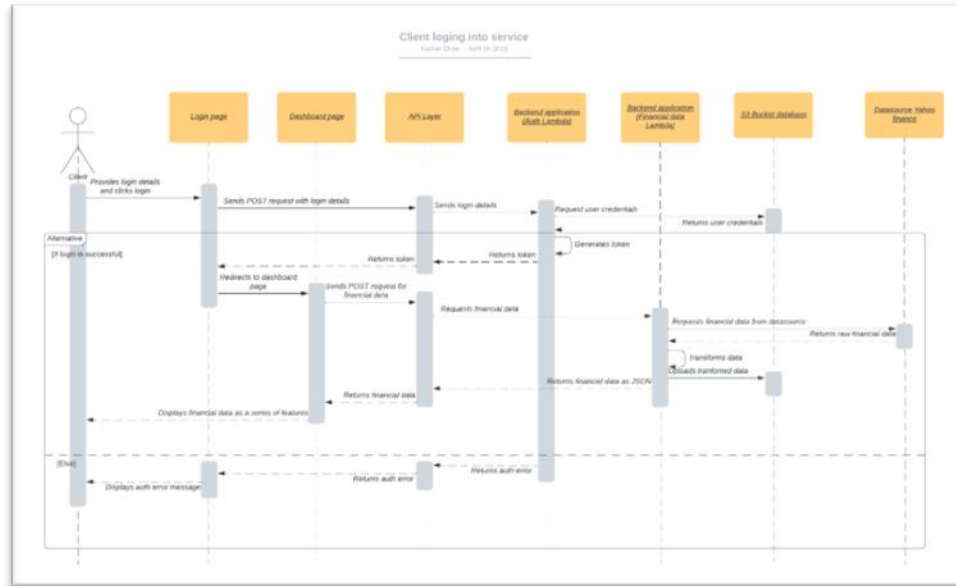


Figure 5.5.1: Sequence Diagram of User Login

Application flow:

- Initially, the user is logged out. (Start)
- The user provides login details and clicks login.
- The request sends login details to the backend lambda.
- Backend lambda checks the credentials of the login.
 - If successful generates and returns a token to the front end.
 - Else returns an error message to the front end. (End)
- Redirects the page to the dashboard.
- Sends post requests to retrieve financial data to the backend.
- The backend returns financial data to the front-end application.
- The front-end application displays financial data to the user. (End)

5.5.2 User is logged in and is viewing financial news data for a specific company

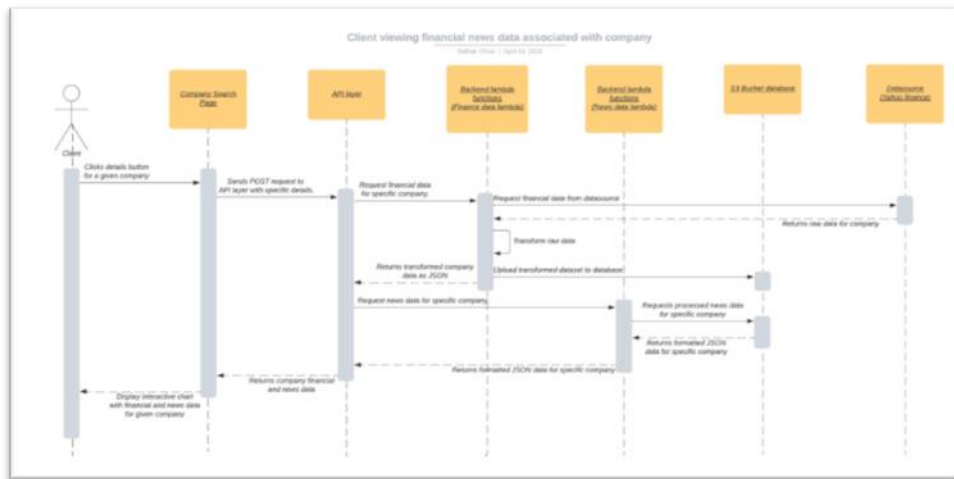


Figure 5.5.2: Sequence Diagram of User Viewing Financial News Data

Application flow:

- The user is initially logged in.
- The user clicks on the details button.
- Request for financial data is sent to backend lambda.
- Backend lambda scrapes and fetches financial data from the data source.
- Financial data is transformed and uploaded to the data lake.
- Transformed financial data is returned to the front end.
- Request for news data is sent to backend lambda.
- Lambda retrieves news data from the S3 bucket.
- News data is returned to the front end.
- Financial and news data is displayed to the user as an interactive chart.

5.6 Web Application Design

This is a link to our deployed frontend application:

https://afzpv4n13.execute-api.ap-southeast-2.amazonaws.com/F12A_ZULU/

5.6.1 UI/UX Considerations

The team has used Tailwind CSS as the core component library for the services frontend application. This allowed the team to produce features and consistent components which make the user experience simple and easy to understand. This includes consistency with fonts and the composition of components displayed on each screen. The team has opted for a purple and white colour scheme to provide a high level of contrast to increase accessibility to users with visual impairments relating to colour blindness. Icons and symbols have also been used to enhance users' understanding of important buttons and features, which will improve the user experience.

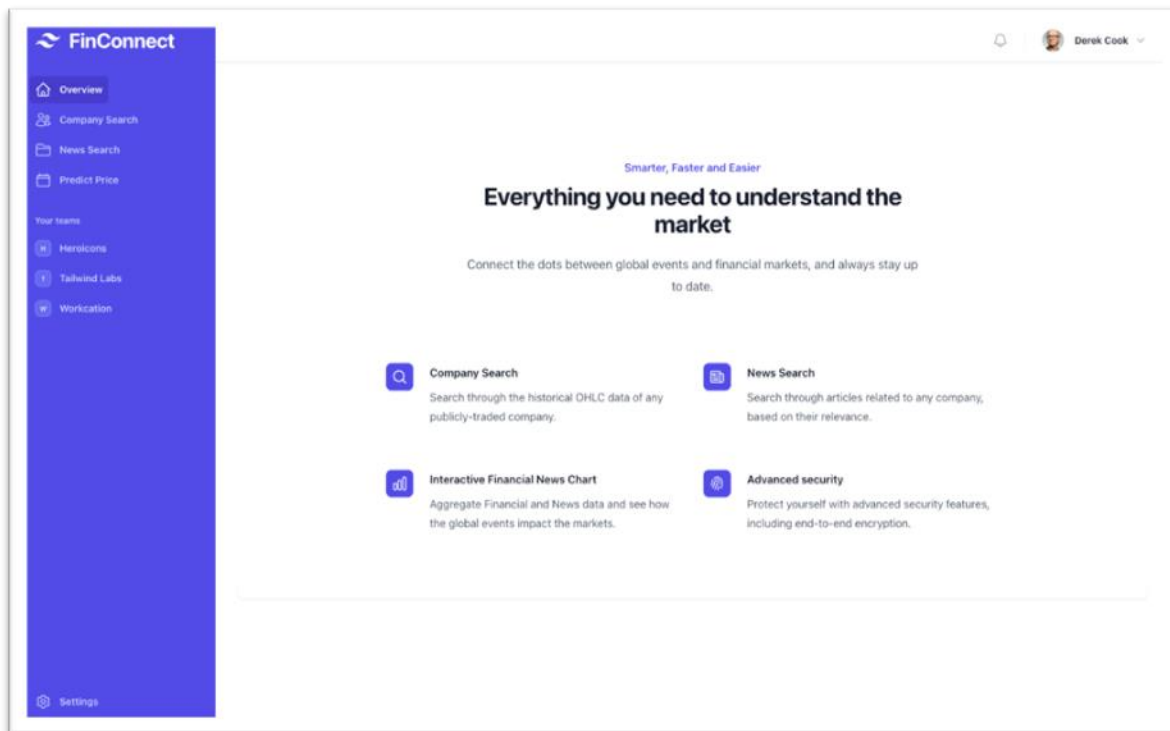


Figure 5.6.1a: Overview page of the FinConnect platform.

FinConnect

Overview
Company Search
News Search
Predict Price
Settings

Your teams
Herolicons
Tailwind Labs
Workcation

Company Search

Search company by company code

Company Code	Open	Close	High	Low	Daily Change	Weekly Change	Monthly Change	Details
AAPL	165.050	165.020	166.450	164.490	-1.630	-0.190	6.090	Details
TSLA	164.800	165.080	166.000	161.320	2.090	-19.920	-27.140	Details
BRK-B	322.360	324.330	324.850	321.610	0.510	4.590	25.960	Details
GOOG	106.090	105.910	106.640	105.485	0.010	-3.550	-0.350	Details
ZZZZ.SR	33.100	34.650	34.750	33.100	1.550	2.050	3.550	Details
JNJ	164.440	162.690	164.970	161.970	-0.890	-3.150	11.560	Details
AMZN	106.100	106.960	108.150	105.080	3.150	4.450	8.250	Details
META	210.210	212.890	213.410	209.580	-0.180	-8.600	8.610	Details
MSFT	285.010	285.760	286.270	283.060	-0.350	-0.380	8.100	Details
NVDA	269.520	271.190	271.830	267.220	0.150	3.610	-0.720	Details

FinConnect

Overview
Company Search
News Search
Predict Price
Settings

Your teams
Herolicons
Tailwind Labs
Workcation

News Search

Search AAPL

"I didn't give permission": Do AI's backers care about data law breaches?	Apr 10, 2023	Link
TechScape: How Nintendo's stayed the most innovative tech company of our time	Feb 14, 2023	Link
Apple posts first revenue drop in four years	Feb 3, 2023	Link
Apple's Tim Cook to take 50% pay hit after shareholder feedback	Jan 13, 2023	Link
TechScape: With a \$67bn takeover in the works, is it finally game on for Microsoft?	Jan 3, 2023	Link
Facebook owner Meta to sack 11,000 workers after revenue collapse	Nov 9, 2022	Link
Will plunging shares end big tech's era of 'pornographic' profits?	Oct 30, 2022	Link
\$80bn wiped from value of Facebook and Instagram owner Meta	Oct 28, 2022	Link
'Bully in a cheap suit': Apple agrees to negotiate with Australian staff after union showdown	Sep 21, 2022	Link
TechScape: The unbearable sadness of Mark Zuckerberg	Aug 31, 2022	Link
Shares in Snapchat owner slump 25% amid slowdown in ad revenue	Jul 22, 2022	Link
'Unstoppable until they aren't': are tech market losses signs of a bust?	May 15, 2022	Link
Meta investors brace for a difficult quarter after stocks nosedive	Apr 24, 2022	Link

Figure 5.6.1b: Company Search page of the FinConnect platform.

Figure 5.6.1c: News Search page of the FinConnect platform.

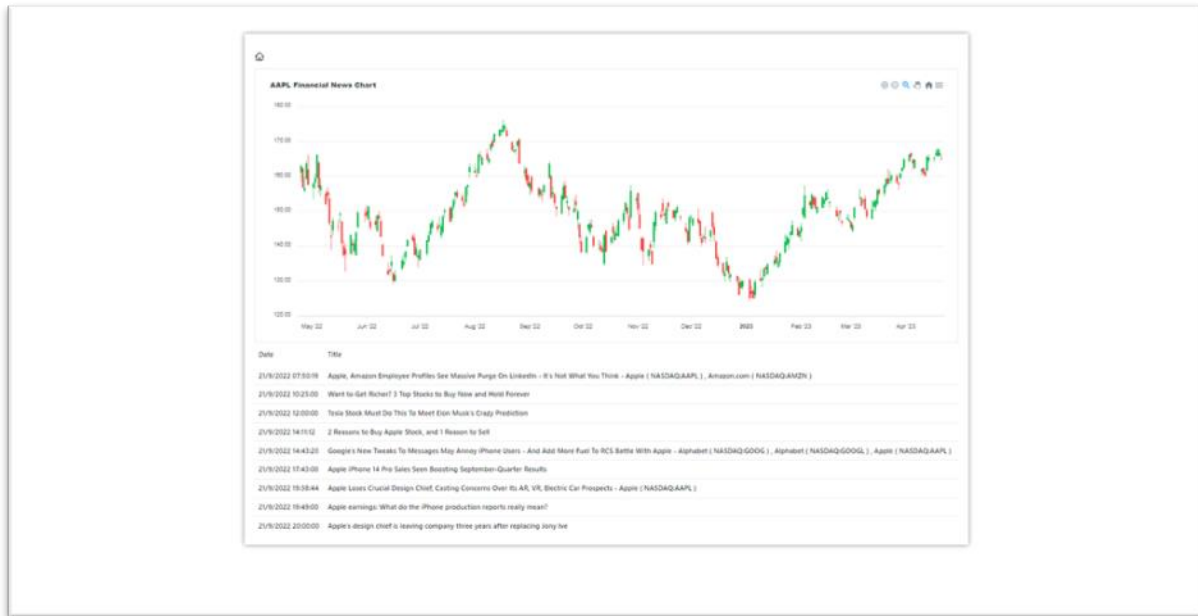


Figure 5.6.1d: Interactive Chart page of the FinConnect platform.

5.6.2 Authentication

The team has implemented authentication into the application. This has been done by utilising the course's authentication lambdas. Users have generated a unique token which is then used to make requests to the backend services which exist behind the course AWS API Gateway. This token is then stored locally for the user.

```
export default function LoginPage() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  function submitLoginForm(e) {
    e.preventDefault();
    const data = {
      username: email,
      password: password,
      group: "F12A-ZULU"
    };

    const config = {
      method: 'post',
      url: '/login',
      headers: {
        'Content-Type': 'application/json',
      },
      data,
    };

    axios.request(config)
      .then((response) => {
        console.log(JSON.stringify(response.data));
        localStorage.setItem('token', response.data.token);
        navigate('/home');
      })
      .catch((error) => {
        alert('Invalid email or password');
      });
  }
}
```

Figure 5.6.2: Login Authentication Request.

5.6.3 Communication with Backend

To communicate with APIs the team has used the Axios package. This allows the frontend application to make POST requests to backend APIs.

```
function searchForOHLC() {  
  let now = new Date();  
  let day = now.getUTCDate().toString().padStart(2, '0');  
  let month = (now.getUTCMonth() + 1).toString().padStart(2, '0'); // January is 0  
  let year = (now.getUTCFullYear() - 1).toString();  
  let date = `${day}/${month}/${year}`;  
  
  const data = {  
    company_ticker: companyTicker,  
    start_date: date  
  };  
  
  let config = {  
    method: 'post',  
    url: '/F12A_ZULU/fetch',  
    headers: {  
      'Authorization': localStorage.getItem('token'),  
      'Content-Type': 'application/json',  
    },  
    data: data  
  };  
  
  axios.request(config)  
    .then((response) => {  
      handleFinancialData(response.data.events)  
    })  
    .catch((error) => {  
      console.log(error);  
    });  
}
```

Figure 5.6.3: OHLC Data Fetch Request.

5.6.4 Frontend Testing

The team has also conducted testing for the front-end application. This was done using the testing framework Cypress.

```

nathanchow@Nathans-MBP F12A_ZULU_FINCONNECT % npx cypress run
Couldn't find tsconfig.json. tsconfig-paths will be skipped

=====

(Run Starting)

Cypress:      12.10.0
Browser:      Electron 106 (headless)
Node Version: v18.12.1 (/usr/local/bin/node)
Specs:        1 found (userflow.cy.js)
Searched:     cypress/e2e/**/*.cy.{js,jsx,ts,tsx}

Running: userflow.cy.js (1 of 1)

logging in
  ✓ should navigate to the login screen successfully (561ms)
  ✓ should navigate to the home page successfully after logging in (1064ms)

signing out
  ✓ should sign out when clicking Sign Out (1447ms)

navigation
  ✓ should go to news search tab (1171ms)
  ✓ should go to company search tab (994ms)
  ✓ should go to predict price tab (1050ms)

searching news and company
  ✓ news search should search aapl and display aapl in search drop down (1401ms)
  ✓ company search should search tsla and display tsla in the list (2183ms)

accessing charts from overview and company search
  ✓ should see charts from details button on company search (2165ms)

9 passing (12s)

(Results)

Tests:      9
Passing:    9
Failing:    0
Pending:    0
Skipped:    0
Screenshots: 0
Video:      true
Duration:   12 seconds
Spec Ran:   userflow.cy.js

(Video)
- Started processing: Compressing to 32 CRF
- Finished processing: 1 second

```

Figure 5.6.4: Cypress Front-end Testing Report.

After completing the test, a video will be generated to playback the test performed live. An example of such a video has been linked here: <https://youtu.be/gFC6NP8FT7Y>.

5.6.5 Deployment

The service has been deployed on an S3 bucket as a statically hosted website. The team has set up a GitHub workflow to build and upload the frontend application to the team's S3 bucket. This setup can be found in the CI.yml file.

```
name: Build and Deploy

on: [push]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repo
        uses: actions/checkout@v2

      - name: Install Dependencies
        run: npm install

      - name: Build
        run: npm run build

      # - name: Run Cypress Tests
      #   uses: cypress-io/github-action@v5
      #   with:
      #     start: npm start

      - name: Upload to S3
        uses: jakejarvis/s3-sync-action@master
        with:
          args: --acl public-read --delete
        env:
          AWS_S3_BUCKET: "f12a-zulu-frontend-s3-bucket"
          AWS_ACCESS_KEY_ID: "${{ secrets.EXEC_AWS_ACCESS_KEY_ID }}"
          AWS_SECRET_ACCESS_KEY: "${{ secrets.EXEC_AWS_SECRET_ACCESS_KEY }}"
          SOURCE_DIR: 'build'
```

Figure 5.6.5: GitHub Build and Deploy Workflow.

The bucket in which the front end was allocated using Terraform. A resource has also been allocated to statically host the website.

```

resource "aws_s3_bucket_website_configuration" "website" {
  bucket = aws_s3_bucket.s3_bucket.id

  index_document {
    suffix = "index.html"
  }
  error_document {
    key = "index.html"
  }
}

resource "aws_s3_bucket" "s3_bucket" {
  bucket = "f12a-zulu-frontend-s3-bucket"
  acl    = "public-read"

  cors_rule {
    allowed_headers = ["Authorization", "Content-Length"]
    allowed_methods = ["GET", "POST"]
    allowed_origins = ["*"]
    max_age_seconds = 3000
  }
}

```

Figure 5.6.6a: Allocation of S3 bucket to statically hosted frontend application.

To properly route the newly created S3 bucket to the course API Gateway, an HTTP proxy was also allocated for the S3 bucket. This takes the URI of the S3 bucket's endpoint and allows it to be proxied to an endpoint that it has been routed to. This integration is then routed to the API Gateway by allocating a unique API Gateway route for the frontend service under the route key `/F12A_ZULU/{proxy+}`.

```

resource "aws_apigatewayv2_integration" "proxy_integration" {
  api_id           = var.gateway_api_id
  integration_type  = "HTTP_PROXY"
  integration_uri   = "http://${aws_s3_bucket_website_configuration.website.website_endpoint}/{proxy}"
  integration_method = "ANY"
}

resource "aws_apigatewayv2_route" "proxy_route" {
  api_id = var.gateway_api_id
  route_key = "ANY /F12A_ZULU/{proxy+}"

  target = "integrations/${aws_apigatewayv2_integration.proxy_integration.id}"
}

```

Figure 5.6.6b: Creating proxy to route S3 static website to the API Gateway.

5.7 Future Design Plans

5.7.1 Caching

In the future, the team plans on implementing caching in the service. Unfortunately, due to time constraints and the team's technical ability, we were unable to complete the implementation of caching within the time frame of the project. Caching in the database will improve data access speeds and overall improve the response time of the service which is a key requirement of our service. The plans for caching have been outlined in the Sprint 3 Vitality Initiative X-factor confluence page as follows: [3. Vitality Initiative \(X-Factor\) Proposal](#).

Along with caching at the database level, the team has also discussed the idea of implementing client-side caching to the web application. This would significantly improve the responsiveness of the web application and would allow for the team's interactive charts to be more performant especially once when charts are constructed for the entirety of a stock's historical data. Due to time constraints and the team's experience with caching, however, no further planning was made for this feature beyond the team's discussion.

5.7.2 Price Prediction with Machine Learning

The team originally planned on incorporating machine learning into the service to generate price predictions for stocks based on news headlines, however, due to the other mandatory requirements required by the course, the team was not able to deliver on this feature. The original plan was to train a natural language processing (NLP) model to perform sentiment analysis on news articles, and then use a regression model to create a price prediction model based on the data gathered from the sentiment analysis. To do this the team would use automated machine learning services like AWS Sage Maker, which provides pre-built NLP algorithms and regression models, which would be trained against news and financial datasets uploaded to the S3 data lake. Following this, once the model would have been trained, a lambda placed behind the courses API Gateway would be deployed, creating an endpoint by which the service could now use to access the model and predict stock prices based on news articles. This plan never came to fruition and the design discussed here has not been formally documented by the team and has only been discussed during the early planning of the service.

6. Maintenance Strategy

6.1 Observability Strategy

6.1.1 Observability data types

The essential data types for observability are **Metrics**, **Events** and **Logs**. These core data types provide us as developers insight into the performance and health of our service as well as any dependencies within our service. By tracking these details about our service, we can identify any undefined behaviour, constraints, and bottlenecks of our service. This will allow the team to more efficiently allocate resources to tasks and overall improve the speed at which we can fix and develop features for the service.

Metrics: The team has set up metrics to monitor how the service is being used. These metrics have been generated with the use of the data type **Events** and will assist in identifying popular calls, and errors and tracking if package dependencies are functioning as intended.

Events: Identify individual user transactions with the service, by indicating the status of an exchange being either a success or an error. Furthermore, events have also been used to gather information about the most used companies which will provide important information to the team when deciding which companies to keep up to date with our scheduled lambda. These events will assist in identifying where errors are in the system and will be used to generate **Metrics** and **Alerts**.

Logs: Keep a record of the transaction details that a user has requested from the service and the output of the service. This will assist in identifying where the error is in the system and what the error is. Logs have been placed to monitor code blocks which can throw errors with detailed error messages, and where possible the exact error output. These logs can then be monitored on CloudWatch.

6.1.2 Three-phase strategy

With these core data types, we must also discuss how we will use this data to properly improve our service. This will be done using a three-phase strategy which consists of firstly monitoring the core elements of data we are collecting continuously as the service operates, secondly incorporating alerts for when abnormal behaviour in our system arises, and thirdly data-driven decision making where we will analyse and identify inefficiencies which will assist in adjusting features and influencing the creation of new features for the service.

Phase 1: Monitoring

Aim: Identify errors and inefficiencies continuously to allow for continuous improvement.

Using the data that we gather, we will be able to swiftly identify where problems in our system occur. This will allow the team to fine-tune the components of our system which will need

adjustment. Once identified, the team can then judge and effectively allocate resources to these components for faster development. This will overall improve the maintainability of the service.

Phase 2: Alerts

Aim: Identify critical errors in the system which must be addressed with urgency.

As the team begins to add and develop new features to the service it can be difficult to identify where things are going wrong. Alerts make identifying the root cause of critical failure fast and easy to detect. This will allow the team to address issues before they reach the entire user base and overall will minimise the number of users who may be affected by errors from development.

Phase 3: Data analytics

Aim: Inform decision-making for resource management and development of future features.

With the data that has been collected, the team will be able to decide which features may not be being used as intended and which features aren't performing as intended. These insights can give an overview of how features may be developed in future or how existing features may be improved or adjusted to customer preferences. Analysing the data collected will be essential to the speed at which the team can improve the service and provide a seamless experience to our users.

6.1.3 Observability Implementation

To incorporate events and metrics, the team has used New Relic, an observability tooling service. New Relic allows us to query our observability data to generate custom metrics and data visualisations in a dashboard. To do this we first needed to integrate New Relic into our service, we needed to link our New Relic account with our lambdas via Terraform. To do this, we injected our New Relic service keys and account details into each Lambdas environment variable and configured New Relic to operate with a serverless client.

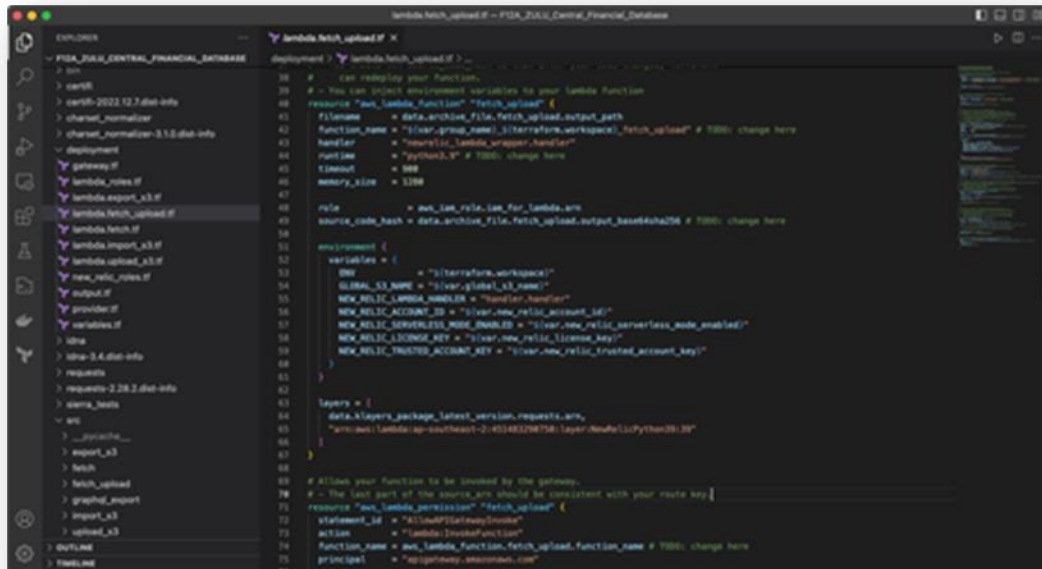


Figure 6.1.3a: Example of New Relic setup in Terraform for the fetch_upload lambda.

From here we utilise the New Relic Python SDK to produce events for our service. To do this, the team has created custom events where critical points of failure may occur.

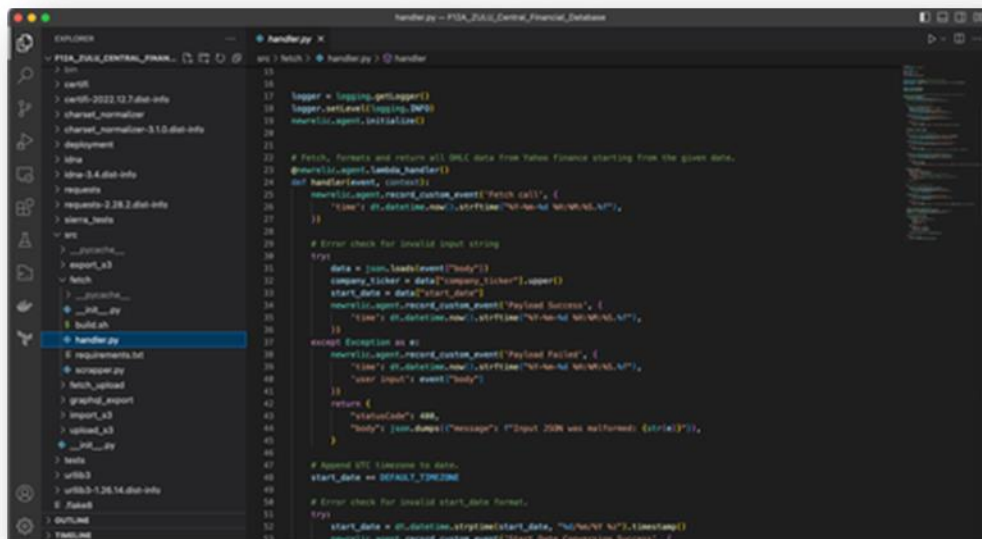
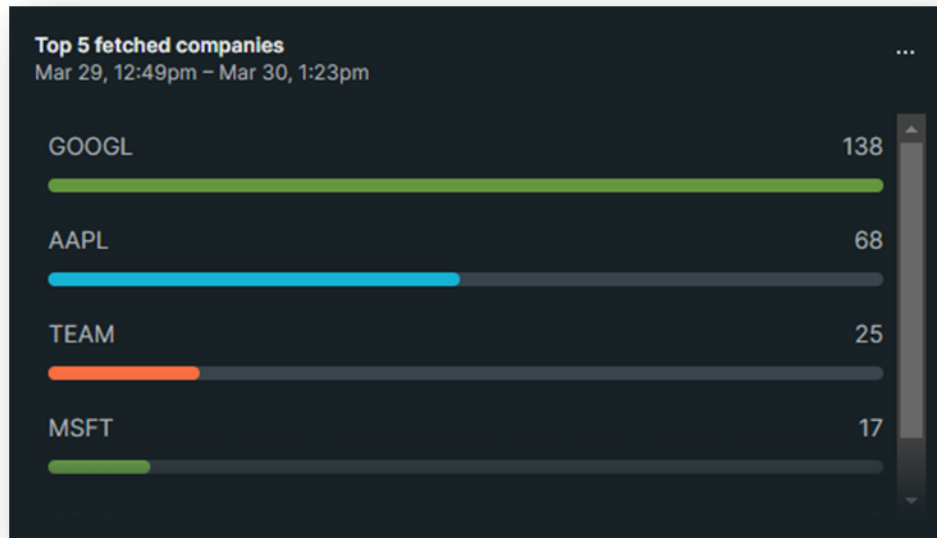
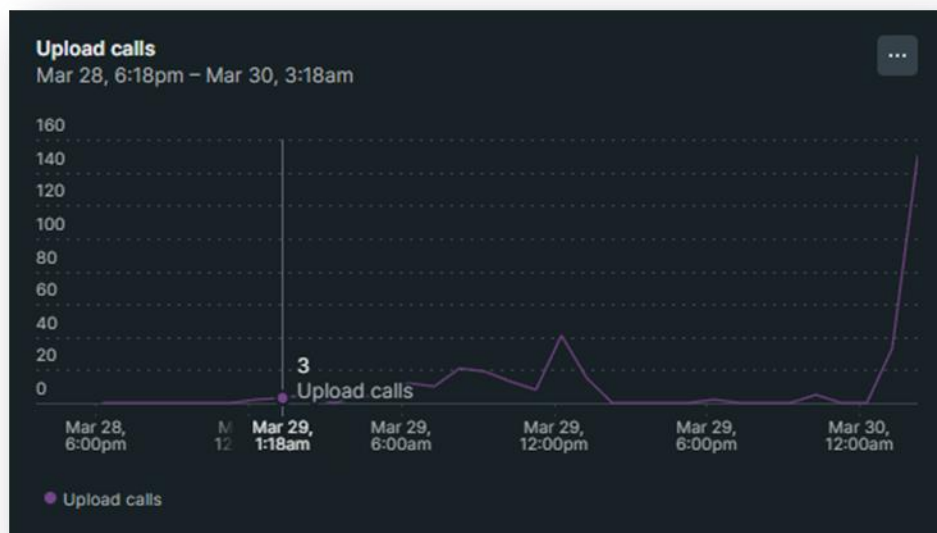


Figure 6.1.3b: Example of custom events with the New Relic Python SDK.

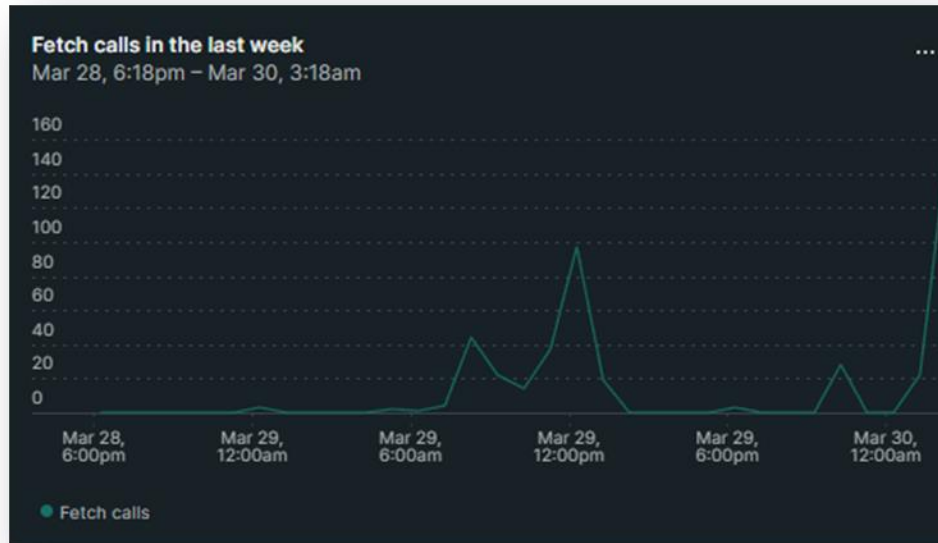
These custom events were used to generate custom metrics and alerts which can be seen below.



Metric 1: Tracking of the top 5 most fetched companies.



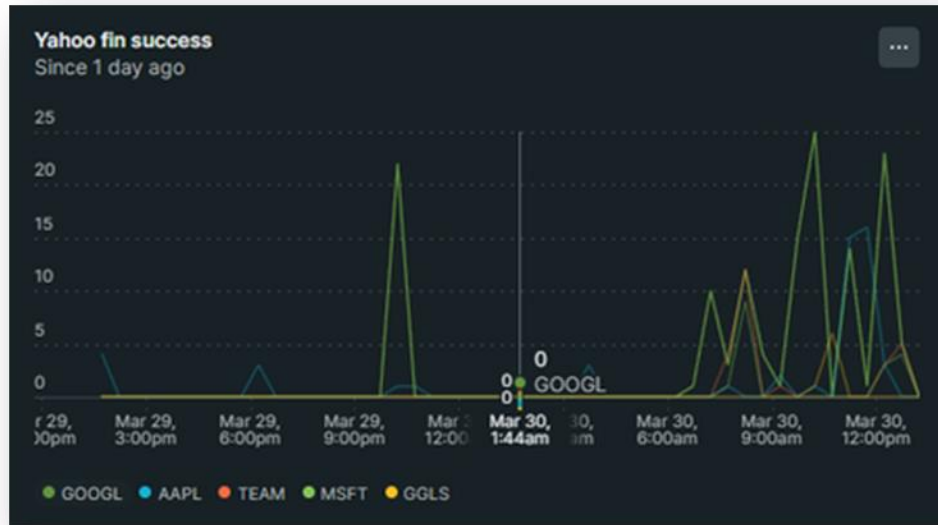
Metric 2: Tracking the number of calls to Upload.



Metric 3: Tracking the number of calls to Fetch.



Metric 4: Tracking the number of failures caused by the Yahoo Fin package.



Metric 5: Tracking the number of successful calls to the Yahoo Fin package.

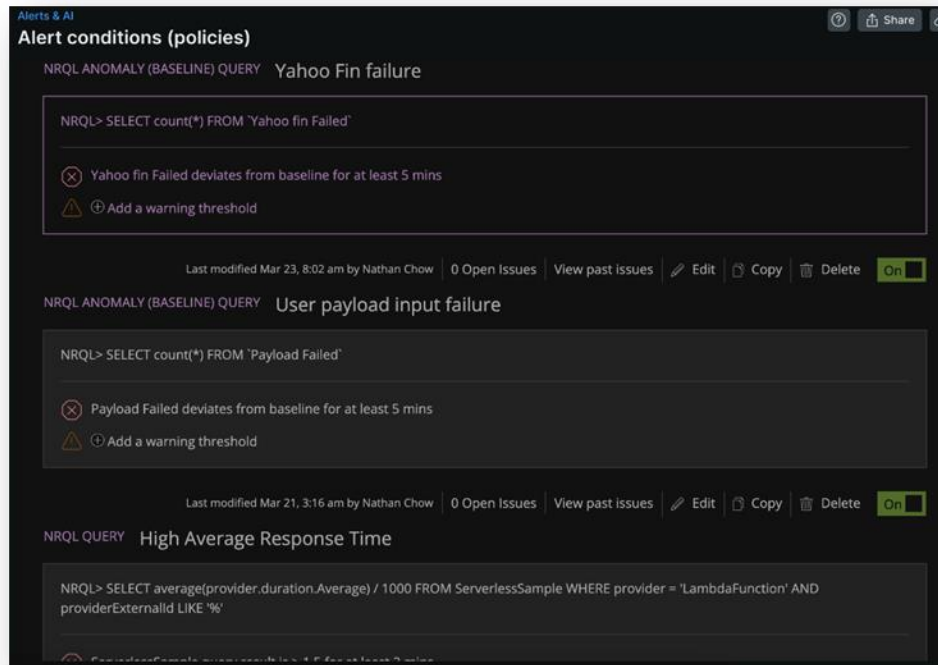


Figure 6.1.3c: An example of a NewRelic alert.

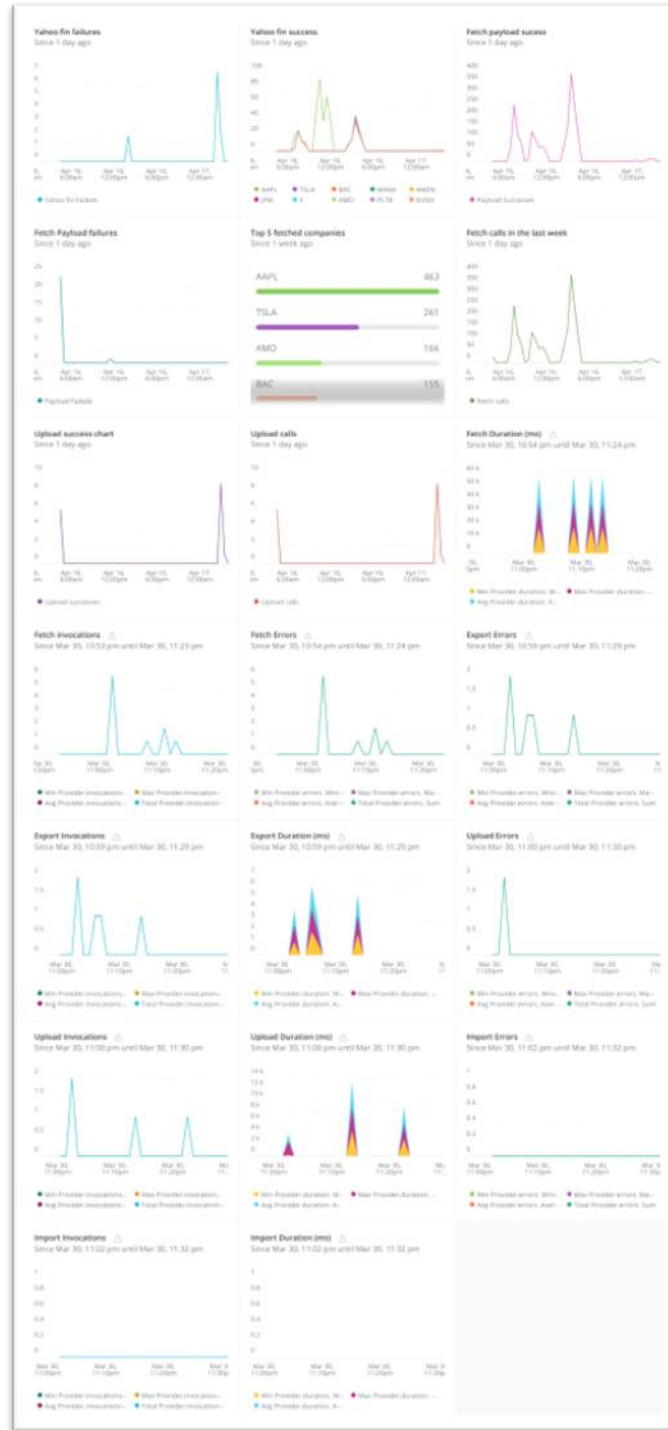


Figure 6.1.3d: NewRelic Team Dashboard.

6.2 Deployment Strategy

To fulfil the deployment to the staging and production environment the following delivery strategy has been defined:

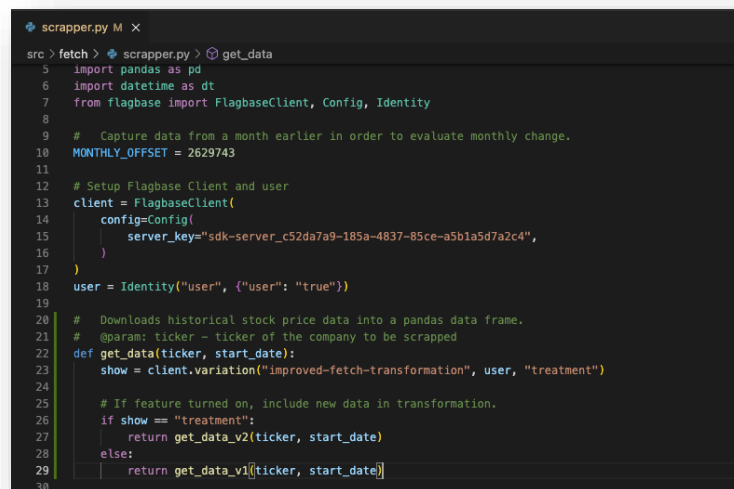
- 1) Setup Observability Metrics - Ensures that all testing on deployment branches is monitored correctly.
- 2) Setup an Experimental Feature Flag for the new feature - In our case, the `get_data_v2` is the new feature.
- 3) Implement feature in the correct format - Ensuring the old feature is backwards compatible.
- 4) Deploy to the dev environment and debug.
 - a) Complete E2E and integration testing from other teams via the CI pipeline.
- 5) Deploy to the staging environment and complete soak testing for at least an hour.
 - a) Utilise the same E2E and integration testing from other teams via the CI pipeline.
- 6) Once soak testing is complete deploy to the production environment.

For the current delivery strategy note that:

1. The NewRelics dashboard will be monitored by Nathan and Kyle.
2. Each rollout, from staging to production should take between 1 hour and 1 day, including soaking.

6.2.1 Feature Flag Implementation

To implement our feature flags, the team has used Flagbase. By utilising the Flagbase python SDK, we were able to connect our Flagbase account to our service utilising a server key. From here we



```

src > fetch > scraper.py > get_data
5 import pandas as pd
6 import datetime as dt
7 from flagbase import FlagbaseClient, Config, Identity
8
9 # Capture data from a month earlier in order to evaluate monthly change.
10 MONTHLY_OFFSET = 2629743
11
12 # Setup Flagbase Client and user
13 client = FlagbaseClient(
14     config=Config(
15         server_key="sdk-server_c52da7a9-185a-4837-85ce-a5b1a5d7a2c4",
16     )
17 )
18 user = Identity("user", {"user": "true"})
19
20 # Downloads historical stock price data into a pandas data frame.
21 # @param: ticker - ticker of the company to be scrapped
22 def get_data(ticker, start_date):
23     show = client.variation("improved-fetch-transformation", user, "treatment")
24
25     # If feature turned on, include new data in transformation.
26     if show == "treatment":
27         return get_data_v2(ticker, start_date)
28     else:
29         return get_data_v1(ticker, start_date)
30
  
```

were able to connect a Flagbase client to our service and configure a feature flag for our fetch service.

Figure 6.2.1: Using the Flagbase python SDK to produce a feature flag for our web scrapper.

6.2.2 Our Flag

As part of the Fetch API transform update, an Experimental feature flag was added to offer backward compatibility for users of the old Fetch whilst allowing A/B testing to be done on the new iteration. The flag is given [here](#) and is used to switch the flag on or off, ensuring correctness once it is deployed to production.

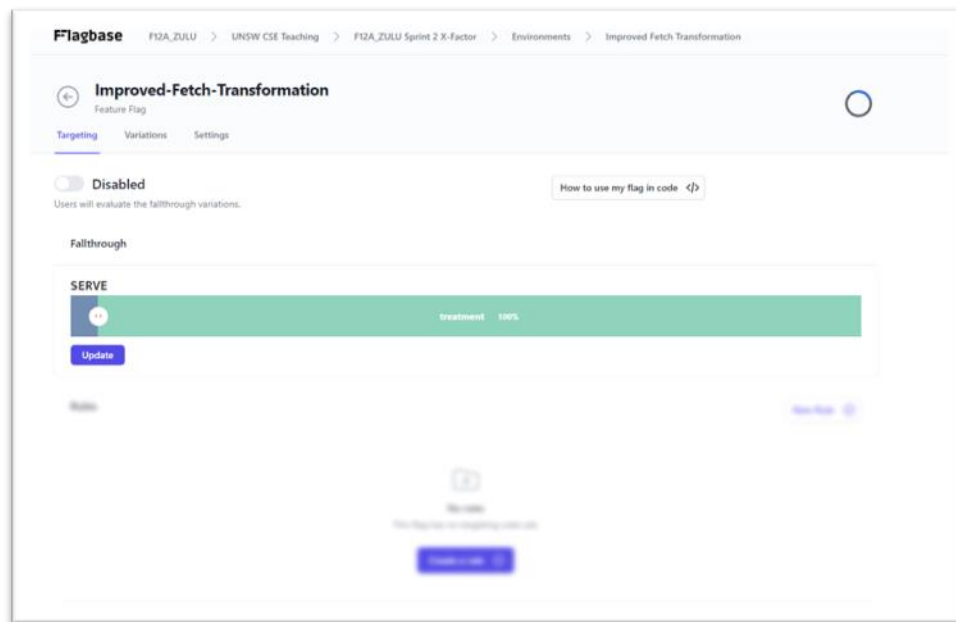



Figure 6.2.2: Flagbase flag used for the introduction of the new feature.

7. Testing

7.1 Our Microservice Testing

Throughout the process of developing our microservice, we utilised continuous integration techniques to ensure that our service was functioning as intended. This involved setting up a GitHub workflow, constructed by the `CI.yml` file in our backend repository, which would perform a predefined list of tests whenever updates were pushed to the codebase. The test performed mainly included black box unit testing and integration testing between individual components of our system. Additionally, team F14A_SIERRA, performed end-to-end (E2E) tests upon our request, which supplemented the high-level testing required to satisfy a complete testing pyramid. Following on from F14A_SIERRA's test, we found that additional E2E tests were required, due to the development of new features by the team. These E2E tests allowed the team to ensure that the service could be properly integrated between components of our service and other services, ensuring a correct user workflow.

To validate the functionality of the team's code and follow the principles of test-driven development, each feature implementation began with the creation of unit tests. These tests were made to verify that any code produced correctly matched the functionality the team set out to achieve and that changes and new features added to the codebase would not break existing code.



```
# Test case 1 - Valid ticker symbol
def test_existing_get_data_success():
    ticker = "AAPL"
    start_date = 946684800 # 1/1/2000
    result = get_data_v1(ticker, start_date)
    assert isinstance(result, pd.DataFrame)
    assert result.index[0].year == 2000
    assert result.index[-1].year <= pd.Timestamp("now").year
    assert result.shape[1] == 6
    assert result.columns.tolist() == [
        "open", "high", "low", "close", "volume", "ticker"]

# Test case 2 - Invalid ticker symbol
def test_existing_get_data_failure():
    ticker = "INVALID"
    start_date = 946684800
    with raises(AssertionError):
        get_data_v1(ticker, start_date)
```

Figure 7.1a: Unit tests are written for the scrapper.

In communication with F14A_SIERRA, the team had been provided with a series of E2E tests. These E2E tests were integrated into the team's pipeline through the GitHub workflow provided by F14A_SIERRA, in the form of a YAML file and Python script.

```

name: "test ZULU CI"

on:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python3
        uses: actions/setup-python@v3
        with:
          python-version: "3.x"
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip3 install -r 'sierra_tests/requirements.txt'
          pip3 install rst2pdf
      - name: Run Python script and save output to file
        run: |
          python3 sierra_tests/sierra_script.py > output.txt
      - name: Convert TXT to PDF
        run: |
          rst2pdf output.txt -o output.pdf
      - name: Upload artifact
        uses: actions/upload-artifact@v2
        with:
          name: output
          path: output.pdf

```

Figure 7.1b: F14A_SIERRA GitHub workflow YML file.

As more features and functionality were added to the service, additional unit tests and integration tests needed to be performed. The team developed these tests and added them to the CI pipeline.

```

37
38 def test_invalid_input():
39     JSON = {
40         "company_ticker": "INVALID",
41         "start_date": "1/1/2019"
42     }
43     response = requests.post(FETCH_UPLOAD, json=JSON, headers=headers)
44     assert response.status_code != 200
45     # If fetch fails, upload also fails - returns an error
46
47
48 # Test checks if the handler returns an error response for a valid fetch request but an invalid upload request
49 # by replicating what the handler does
50 def test_upload_only_fail():
51     JSON = {
52         "company_ticker": "AAPL",
53         "start_date": "1/1/2019"
54     }
55     response = requests.post(FETCH, json=JSON, headers=headers)
56     return_values = response.json()
57
58     # remove dataset_id to cause an error when uploading
59     remove_dataset_id = return_values.pop("dataset_id", None)
60     response_update = requests.post(UPLOAD, json=remove_dataset_id, headers=headers)
61     assert response_update.status_code != 200

```

Figure 7.1c: Example of teams E2E tests written for the fetch_upload endpoint.

7.2 Testing H09A_FOXTROT's Microservice

In addition to testing our service's endpoints, we also tested another team's microservice, H09A_FOXTROT. The decision to test H09A_FOXTROT was made due to the relevance it would hold in integration with our full vision for our microservice.

The overall goal was to make a microservice that can compare financial data to world events by key terms. Thus, testing and ensuring their service works as expected will benefit us to provide a reliable service on that we can then base our results and make a service that is reliable and can perform optimal progress for our clients.

7.2.1 Justification

We used their /graphql endpoint to test their functions, which included:

- howsItTrending
- dailySummary
- contentSearch
- mostRecentArticles
- relatedArticles

The tests we wrote are **end-to-end(E2E)** tests, as per the request of team H09A_FOXTROT, that check the response of their /graphql endpoint's functions when given queries and graphql variables.

Since we were planning to use their microservice, we decided to make sure that their functions work well and sent correct error messages. We mainly focused on testing their functions because they are a core feature of their microservice, which processes data scraped from The Guardian and the New York Times.

7.2.2 Components

- The deployed API route and all its functions (/graphql).
- Check if correct errors are thrown appropriately for all the functions in /graphql.
- Find more possible gaps in the deployed API and its routes.
- Ensure that there are no potential faults in the functionality we plan to use.
- Ensure the connection to the API is continuous.
- Tests for backwards compatibility of the code base as the application is scaled.
- Provide any information on any potential issues or problems that may be caused along the branch.

7.2.3 Implementation

To incorporate our tests with team H09A Foxtrot's service, the team has requested that we test on their **staging** environment. Our approach was to create a new repository specifically to deploy a new service aimed at testing Foxtrot's service.

For Foxtrot to run our tests, we decided on creating a testing lambda function, which would request the test information via a Python script which would be run via a GitHub action. To do this a YAML file was defined on `workflow_dispatch`, which would run a Python script, calling the endpoint of our deployed lambda. This lambda will hence run our tests and write the results of the tests to a file called `output.txt`. Within the workflow that we have defined, this `output.txt` file is then converted to a PDF file and uploaded as an artefact.

Our tests for each of the individual functions for Foxtrot's service have been separated into their files. Each test checks the output of a request to their service defining some behaviour of their service and comparing it to some expected output. The test function will then return a string indicating "SUCCESS" or "FAILURE" corresponding to whether the test completes successfully. Within the lambda's `handler.py` file, each test is invoked and prints to a string containing which feature is being tested, which test is running and the result of the test. This string is then written to the `output.txt` file to store the results of the test for the GitHub workflow to then convert to PDF and upload as an artefact.

Examples of our tests can be found under the `test` directory from our testing service repo. The link to the tests can be found here:

https://github.com/cseteaching-unsw-edu-au/F12A_ZULU_Testing_Microservice/tree/lambda_deploy/src/test_foxtrot

F12A_ZULU Final Design Report

Instructions were then sent to H09A_Foxtrot on how to incorporate our testing microservice.

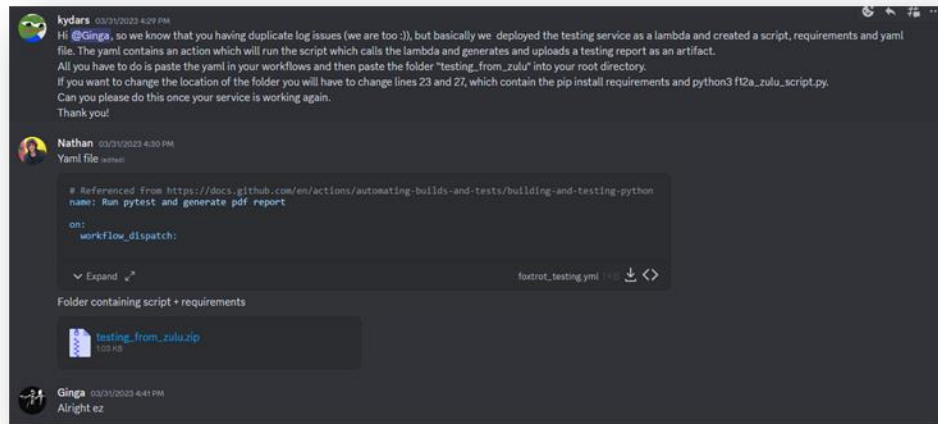


Figure 7.2.3: Instructions on how to incorporate our testing service.

8. Ecosystem Service Integration

8.1 Overview

During the development of our application, we integrated existing services from the company ecosystem into our front end, as a means to avoid redundancy within the company codebase. The services integrated into the end solution included the login and register routes and a news export service created by team H09A_FOXTROT. The login and register routes offered a means of authentication, whilst the news export service was used as a means to fetch relevant news data which could be aggregated with our financial data.

8.2 Justification

The integration of these existing services from the company ecosystem was a strategic decision to leverage existing functionality and resources within the company. By reusing the login and register routes, we were able to implement a secure and consistent authentication process, ensuring the confidentiality and integrity of user data. Additionally, the integration of the news export service allowed us to access up-to-date news data that complemented our financial data, enhancing the overall value of our application. The news service also offered the ability to filter by date range, which would allow our service to reduce cases of over-fetching, thereby optimizing performance.

8.3 Evidence of Use

As shown below, the news export service was integrated by sending a graphql query searching for the name given by the user and then setting the value of the articles attribute to be the processed

```
export function newsSearchCall (name) {
  let data = JSON.stringify({
    query: `
      contentSearch(page:1, pageSize:50, q: {operation: AND, subQueries: [{term: "${name.split(" ")}`)}], (term: "stock")}, filters: {section: "technology"}, orderBy: newest, orderDate: published)
    {
      total
      currentPage
      pageSize
      totalPages
      results {
        webPublicationDate
        sectionId
        sectionName
        suburl
        webTitle
      }
    }
  `
  },
  variables: {},
  });
  let config = {
    method: 'post',
    maxBodyLength: Infinity,
    url: '/news_FOXROT/graphql',
    headers: {
      'Authorization': 'Bearer ' + localStorage.getItem('token'),
      'Content-Type': 'application/json'
    },
    data : data
  };
  axios.request(config)
    .then((response) => {
      const list = filterAndFormatList(response.data.data.contentSearch.results)
      setShowSpinner(false);
      setShowArticles(true);
      setArticles(list);
    })
    .catch((error) => {
      console.log(error);
    });
}
```

response data.

Figure 8.3: Axios request made to news export service to retrieve news data.

9. Challenges and Constraints

9.1 Overview

Through working on FinConnect throughout the three sprints many issues both technical and non-technical arose. From approaching the assignment with the theory of constraints in mind, it is crucial to avoid any bottlenecks, blockers, and obstacles as they eventually build up and stunt the performance of the system the team intended to design. Identifying these blockers early meant more efficient planning and understanding of the issues the team would run into, which would thus mitigate how they would impact the team's performance. This section will touch upon both the technical and non-technical constraints and challenges that the team has faced throughout the development of FinConnect, along with the risk management assessment that the team has devised.

9.2 Technical constraints

9.2.1 Third-party Compatibility and Dependencies

Identifying the constraint

With the creation of a microservice for an ecosystem, dependencies and other microservices and third-party APIs are expected. This can prove to be a constraint as systems are always updated, and new systems are often required to be compatible with legacy systems.

Exploit the constraint

To exploit this constraint, planning and having a clear understanding of the compatibility of different services should be done before. This includes ideas such as avoiding services that are old or aren't frequently updated and communicating with other groups to ensure similarity.

Subordinate the constraint

To subordinate this constraint, services should be checked for updates frequently, and new features should be inspected thoroughly before updating to ensure backward compatibility with other services.

Elevate the constraint

To elevate this constraint, services that are not yet created by third-party developers should be developed ourselves. For example, a financial data scraper has been sourced however a news data scraper hasn't. Therefore, a news data scraper should be implemented while also making sure it is compatible and runs alongside the financial data scraper.

9.2.2 Knowledge and Inexperience

Identifying the constraint

The course ecosystem involves utilising AWS services through programmatic means via Terraform. Along with this, new technologies such as GraphQL, New Relic and Flagbase have

also been required within the development of the microservice. The team has had little to no prior experience with any of these technologies.

Exploit the constraint

As the team is inexperienced, the team will leverage resources such as the course forums, and online documentation and start on development early.

Subordinate the constraint

To subordinate the constraint, the team will meet regularly to attempt to resolve issues with paired programming. Team members have agreed to be openly available for meetings outside the designated team-wide meetings for such purposes.

Elevate the constraint

To elevate the constraint the team will assign members to specific new technologies to specialise in understanding. By having members specialise in specific new technologies, team members can then teach each other the basics of the new technology which they will require when used, rather than having all members perform the same research for each technology.

9.3 Non-Technical Constraints

9.3.1 Data Quality and Integrity

Identifying the constraint

Data that is unreliable can be a severe detriment to microservices that aim to provide useful statistical information. News data is notorious for being unreliable and finding credible sources that are free from bias can prove difficult.

Exploit the constraint

To exploit this constraint, data quality requirements should be established that is based on the purpose of the microservice. These would include conditions such as only obtaining data from credible/relevant news sources and avoiding incomplete data.

Subordinate the constraint

To subordinate the constraint, data quality should be consistently monitored to make sure provided information from the microservice is credible and reliable. To do this, a data maintenance practice that checks for duplicates, errors, and empty values should be adopted.

Elevate the constraint

To elevate the constraint, this data maintenance practice should be automated so that effort can be used elsewhere in expanding and subsequently eliminating the constraint. Data maintenance should also cover if the sources ever update or change, thus the service should be able to notify and do its best to adjust to changes.

9.3.2 Data Scarcity and Abundance

Identifying the constraint

When considering a microservice that connects different data types, the availability of each must be sufficient and comparable. Data scarcity is a permanent constraint and is out of the scope of what a project management team can remedy. However, data abundance is a resource constraint on manageability and can be mitigated.

Exploit the constraint

To exploit this constraint, clear restrictions on the domain of data that is scraped should be set. These would include conditions such as time frame and type of data. Doing this would decrease the amount of data to maintain, but also may affect trend analysis aspects such as comparing vast time frames together.

Subordinate the constraint

To subordinate this constraint, data analytics tools could be used to automate the process of data scraping. These techniques will also aid in identifying patterns and relationships that might have otherwise been missed, as well as making it easier to maintain constant data gathering.

Elevate the constraint

To elevate this constraint, collaboration with groups that specialize in data science may help in ensuring data is identified and interpreted correctly. This will allow even small amounts of data to provide relevant and useful information.

9.3.3 Time Limitations

Identifying the constraint

The entire project must be completed in 10 weeks, with around 3 weeks separating a total of 3 sprints. If not managed well, this constraint can significantly impact the quality of our microservice and there should be avoided.

Exploit the constraint

To exploit this constraint, a sufficiently detailed Jira board should be established at the start. This should include due dates and tasks through the use of scrum boards and road maps.

Subordinate the constraint

To subordinate this constraint, the Jira board should be constantly checked and left flexible to accommodate any issues that may arise during development. This will ensure tasks are done on time and in order and will help in avoiding situations where dependent tasks are bottlenecked by other tasks.

Elevate the constraint

To elevate this constraint, team members should adhere to the due dates and work proactively by posting any issues they encounter on the Jira board. Likewise planning any task dependencies in the program, so that the schedule operates within optimal time is critical to ensuring a lack of

frustration within group members and that tasks can be handed around and the project can progress at the expected time.

9.3.4 Scope Creep

Identifying the constraint

Often in group-based projects, non-agreed-upon features may be implemented due to a lack of requirements management. This is an issue as it deviates from the specification of the project, as well as influences the flow of tasks, creating bottlenecks.

Exploit the constraint

To exploit this constraint, fixed dates for meetings/stand-ups should be adopted. This ensures that there will be frequent communication between the team members, giving opportunities to discuss scope and expectations.

Subordinate the constraint

To subordinate this constraint, meetup times should be kept to a strict schedule and happen twice a week, once in person and another online. This will allow for constant communication, and the strict schedule will ensure this constraint is kept as a primary concern.

Elevate the constraint

To elevate this constraint, team members should communicate outside of meetup times to resolve minor issues/misconceptions. Directly communicating and resolving any issue as soon as possible prevents clashes and egos building within the group. This will expend more time, but also significantly reduce the risk of scope creep by ensuring individuals are doing exactly what their task requires.

9.4 Risk Management

In line with our discussion of the constraints related to the production of our microservice, this section will outline the risks associated with implementation and detail the respective countermeasures we employed to mitigate risks.

Table 9.4a: Risk Matrix

	Trivial	Minor	Moderate	Significant	Extreme
Very Unlikely	Low	Low	Low	Medium	Medium
Unlikely	Low	Low	Medium	Medium	Medium
Possible	Low	Medium	Medium	Medium	High
Likely	Medium	Medium	Medium	High	High
Very Likely	Medium	Medium	High	High	High

Table 9.4b: Risk Levels and Mitigation Techniques

Risks	Details	Level	Mitigation
Failing to meet the required deadline	Due to the inexperience of the team with AWS and Terraform, it is possible that the team may not have the technical skill to produce the product to the desired standard.	Low	Minor impact To minimise the impact of this risk, the team has clearly outlined that the product that is being produced will be fairly minimal, delivering a smaller set of features to a high standard. The team has also discussed clearly which objects are stretch goals which will take a lower priority in the final product.
			Unlikely To reduce the likelihood of this happening, the team has delegated a sub-team to manage and attempt to implement new technologies. This aims to minimise coupling and dependencies within the codebase allowing for parallel development.
Data Security and Privacy	Due to the prevalence of hackers, the platform may experience data breaches or illegal access to data, both of which could have severe repercussions for the users of the microservices.	Medium	Extreme Impact To reduce the potential impact of a data breach, all sensitive data should be heavily encrypted and different types of data should be separated to avoid global access.
			Unlikely To reduce the likelihood of a data breach, the microservice should employ multi-factor authentication as well as routine security audits in order to investigate suspicious activity.

Inaccuracy	Due to unreliable data sources, the microservices may produce inaccurate financial data, which will negatively impact users who depend upon the service to inform their financial decisions.	Medium	Significant Impact Reducing the impact of inaccurate data can be done by monitoring the various sources of data and flagging inaccurate sources.
			Unlikely Reducing the possibility of inaccurate data can be achieved through establishing quality assurance processes such as routinely monitoring data sources and algorithms, as well as directly verifying data against multiple sources.
Permissions and access issues when provisioning resources	It is highly likely that the team will not have the required permissions to provision resources in AWS.	Medium	Moderate impact The impact of begin unable to provision resources on AWS is moderate. If a particular service the team wants to use, but is not given the permission to use, alternatives may exist which could serve the same purpose. Otherwise, it is still within reason to remove the feature.
			Unlikely It is unlikely that this may occur. Most of the services the team will be using will exist within the realm of services the course has provided access to.
Market Volatility	Due to sudden and unpredictable fluctuations in financial data as a result of changes in the market, the	High	Significant Impact The impact of fluctuations in financial data can be mitigated through regular updates to the S3 lake and the identification of market trends (for example, technology companies fluctuate more than banks)

	microservices may misinform users with outdated data, posing a major risk		<p>Likely</p> <p>The likelihood of market volatility solely relies upon the market it exists in and thus can only be mitigated through using a direct notification system, which will allow the user to respond immediately to market changes.</p>
--	---	--	---

10. Conclusion

In conclusion, this report serves as a comprehensive justification for the project, highlighting the findings from market research and the business value of the project. The backend and frontend requirements have been thoroughly analysed and addressed, and the software architecture has been designed according to these requirements. A maintenance strategy has been correctly implemented to ensure ongoing performance and reliability and the integration of another service into the system has been completed. Overall, this report provides a robust justification for the project, demonstrating the thorough planning and consideration of various aspects of FinConnect, which have ensured its successful implementation.

11. References

1. Grand View Research. (2022). E-commerce Market Size, Share & Trends Analysis Report By Type (B2B, B2C), By Application (Electronics & Appliances, Fashion & Apparel, Home & Furniture), By Region, And Segment Forecasts, 2020 - 2027, <https://www.grandviewresearch.com/industry-analysis/e-commerce-market>
2. OECD (2021), Artificial Intelligence, Machine Learning and Big Data in Finance: Opportunities, Challenges, and Implications for Policy Makers, <https://www.oecd.org/finance/artificial-intelligence-machine-learningbig-data-in-finance.htm>.

12. Appendix

- Links to the codebases for the project:
 - [Backend Repository Link](#)
 - [Frontend Repository Link](#)
 - [Testing Microservice Repository Link](#)