8 April 2021

CS 4471B
Software Design and Architecture

Term Project
Progress Report 3: Final Implementation

Service-Oriented Architectures

Group 7

Sam Ahsan
Peter Beaulieu
Nick Gardi
Andrew Greaves

Table of Contents

Section 5: Allocation View

Section 6: Sequence View

Section 7: User Interface State View

1. Preliminary Reading

The goal of this project was to familiarise the team with cloud-based services, architectures, and patterns, and with architectural documentation. The project was developed over the course of 4 weeks. This document is the last of three progress reports and documents the project in its final state of implementation. As of this writing, the “Breadcrumbs” stock tracker, index tracker, and currency trackers are running on the IBM Cloud platform as initially intended. The team’s Kubernetes cluster on IBM Cloud runs a single worker node which provides the network location where the project’s runtime environment is executed. Services are added and removed dynamically using the IBM Cloud CLI with kubectl commands to create or remove service pods. The application running the group’s three proprietary services on the cloud can be accessed at http://169.57.42.50:32506/login. Alternatively, the front end application can be run locally using the commands **npm install** followed by **npm start** in the root directory of the source code.

Several notable changes were made between this document and its predecessor. Some of the outsourced software services that were initially intended to be used for the project were replaced with more suitable alternatives, and some were removed entirely to enforce simplicity in the architecture. These changes are as follows: firstly, the team had projected use of Qt for front-end development during the initial stages of project planning. In this final implementation, the front end was developed using Node.js with React JS frameworks for the result of a simple and user-friendly interface. Secondly, the data API used for the final implementation is as provided by twelvedata, which allows for consistent data requests across all services. Thirdly, IBM EventStreams and the IBM API Gateway were removed from the project architecture, allowing for back-end simplicity and tighter cohesion in the final implementation. IBM Object Storage was replaced with Google FireBase, which also loosened some architectural coupling at the back end of the development work. Also notably, the Notification Layer, Notification Hub, and Notification Bus entities in the previous document (progress report #2) were respectively renamed as Access Layer, Access Hub, and Access Bus, which more accurately describes the functionality of these entities in the final implementation.

Overall, the team was successful throughout the development as projected by the project plan. The discussed changes are reflected in the following documentation.

2. Documentation Beyond Views

2.1: Documentation Roadmap

2.1.1: Scope and Summary

This document contains a comprehensive documentation of the architecture and patterns used to design the system. In this section, document organisation is discussed to point the reader to a given section of the document (the table of contents on the previous page can also be addressed for this purpose). An overview of the architectural views used within the system is presented here, highlighting its elements and relationships. In-depth discussion of each view's elements and relationships is discussed in the view's section (see table of contents). Finally, stakeholders can refer to the end of this section to determine which sections of the document can be used to suit their needs. This document does not contain preliminary documentation, requirements and specifications of the system, development team information, or project timeline information used to initiate development. Such information can be found in the group's previous progress report (Progress Report 1) which has no significant changes to be documented at the time of this writing.

2.1.2: Document Organisation

This document is organised according to seven sections. The first subsection of section two of this document provides a roadmap consisting of a scope and summary of the whole document (see previous subsection), followed by an overview of the views used to represent the system as discussed in this document, then information for specific stakeholders to direct them to a particular point in the document. The next subsection in section one discusses the methods used to document each view discussed, which is elaborated upon by discussing the mapping between each of the views in the following subsection. The last two subsections of section one respectively discuss a rationale for the architecture decisions made throughout this document, as well as a directory for readers to find reference material within this document more quickly.

Sections 3 through 7 of this document each focus on a particular view that was used to represent the system. Each of these sections is broken into five subsections which respectively document a graphical primary presentation, an element catalogue, a context diagram, a variability guide, and finally a rationale pertaining to the particular view that the section is focussed on.

2.1.3: View Overview

Five views are used to develop this system, including three structural views and two quality views. The three structural views include a module view, a component-and-connector view, and an allocation view. The two quality views include <> and <>. These five views, along with their element types, property types, and relationship types are summarised in the table below.

View	Element Types	Property Types	Relationship Types
Module View	- Layers - Sub-modules	- Responsibilities - Visibility of interfaces - Implementation information	- *uses* - *inherits*
C&C View	- Components - Connectors	- Reliability - Performance - Resource requirements - Functionality - Security - Concurrency - Modifiability - Tier	- *uses* - *publishes* - *is-a*
Allocation View	- Software Elements - Environmental Elements	- Reliability - Performance - Resource requirements - Functionality - Security - Concurrency - Modifiability - Tier	- *contains*
Sequence View	- Subscriber - UI - Service Registry - FireBase - twelvedata API	- Binding time - Triggers - Return values - Scope	- *executes* - *returns*
User Interface State View	- Creating new subscriber - Authenticating - Displaying subscriber UI - Managing subscriber services - Retrieving data	- Binding time - Triggers - State changes - Scope	- *goes-to*

2.1.4: For Stakeholders

Developers of this system can consult the primary presentation of each view for a holistic understanding of the elements in the system. Developers can also consult the element catalogue for a more in-depth understanding of the role of each component, module, or entity involved in each view. The element catalogue is also useful for developers interested in quality attributes projected from each view, as well as the relationship properties included in the element catalogue which will help developers and system architects determine dependencies and associations between the given elements.

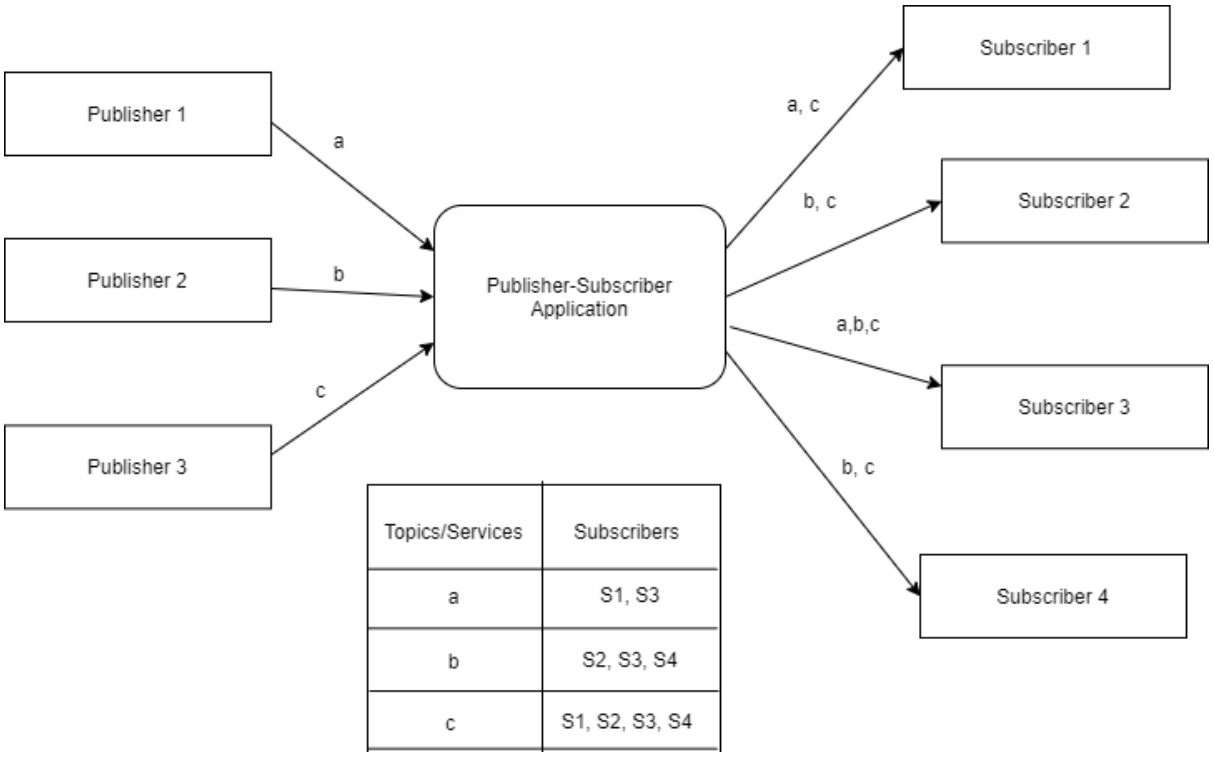
Quality testers and system maintainers will be interested in the context diagrams presented in each view as well as the variability guide which may aid in identifying areas of the system in need of further maintenance or modification.

Business-oriented stakeholders will also find the context diagrams as well as the rationales presented at the end of each view to be useful for determining why certain decisions were made and if any business modifications are possible or valuable.

2.2: How a View is Documented

Each view contained in this documentation is described according to a primary presentation, an event catalogue, a context diagram, a variability guide, and a rationale. The primary presentation for each view is a graphical diagrammatic representation of the system using a distinct set of elements, components, or entities. The event catalogue for each view is divided into two sections which respectively document the elements and their properties in the view, and the relations and their properties in the view extended beyond the information available in the primary presentation. The context diagram is a second diagrammatic representation of the view and gives further context to the elements, components, or entities as a system existing in the physical world. The variability guide for each view describes options and decisions that were considered while architecting the system. Finally, the rationale provides a short discussion on why the given view was chosen to represent the system and how each view can be used to reason about various attributes of the system.

2.3: System Overview



Topics/Services	Subscribers
a	S1, S3
b	S2, S3, S4
c	S1, S2, S3, S4

This system uses a Broadcast-based Publish-Subscribe Pattern (see generic SOA pub-sub architecture diagram above). For this application, the system supports three publishers as per the services offered by the application, and a dynamic number of registered subscribers. Publishers are not required to be aware of the subscribers in the system since the publishers broadcast updates to the access hub/event bus.

2.4: Mapping Between Views

Element in Module View	Elements in Other Views	Mapping Relationship
Services Layer	- Services Module (C&C View) - Services Container (Allocation View)	- *is* (same element) - *is* (same element)
Access Hub	- Access Bus (C&C View)	- *is* (same element)
Google FireBase	- FireBase (C&C View)	- *is* (same element)

	- FireBase (Allocation View) - FireBase (Sequence View) - twelvedata API (Sequence View) - Retrieving data (State View)	- *is* (same element) - *is* (same element) - *implemented by* - *occurs while*
Service Registry	- Service Registry (C&C View) - Service Registry (Sequence View) - Managing subscriber services (Sequence View)	- *is* (same element) - *is* (same element) - *occurs while*
Subscriber Layer	- Front End (C&C View) - Front End, UI (Allocation View) - UI (Sequence View) - Creating subscriber, Authenticating, Displaying UI (Sequence View)	- *is* (same element) - *is* (same element), *implemented by* - *implemented by* - *implemented by*
Subscription Manager	- Subscription Manager (Allocation View) - Managing services (State View)	- *is* (same element) - *implemented by*

2.5: Rationale

The architectural decisions developed for this project are adapted for a service-oriented architecture deployed using IBM Cloud Services. The five views used to describe the system follow a publish-subscribe methodology because it enables event-driven architectures and asynchronous parallel processing, while improving performance, reliability and scalability.

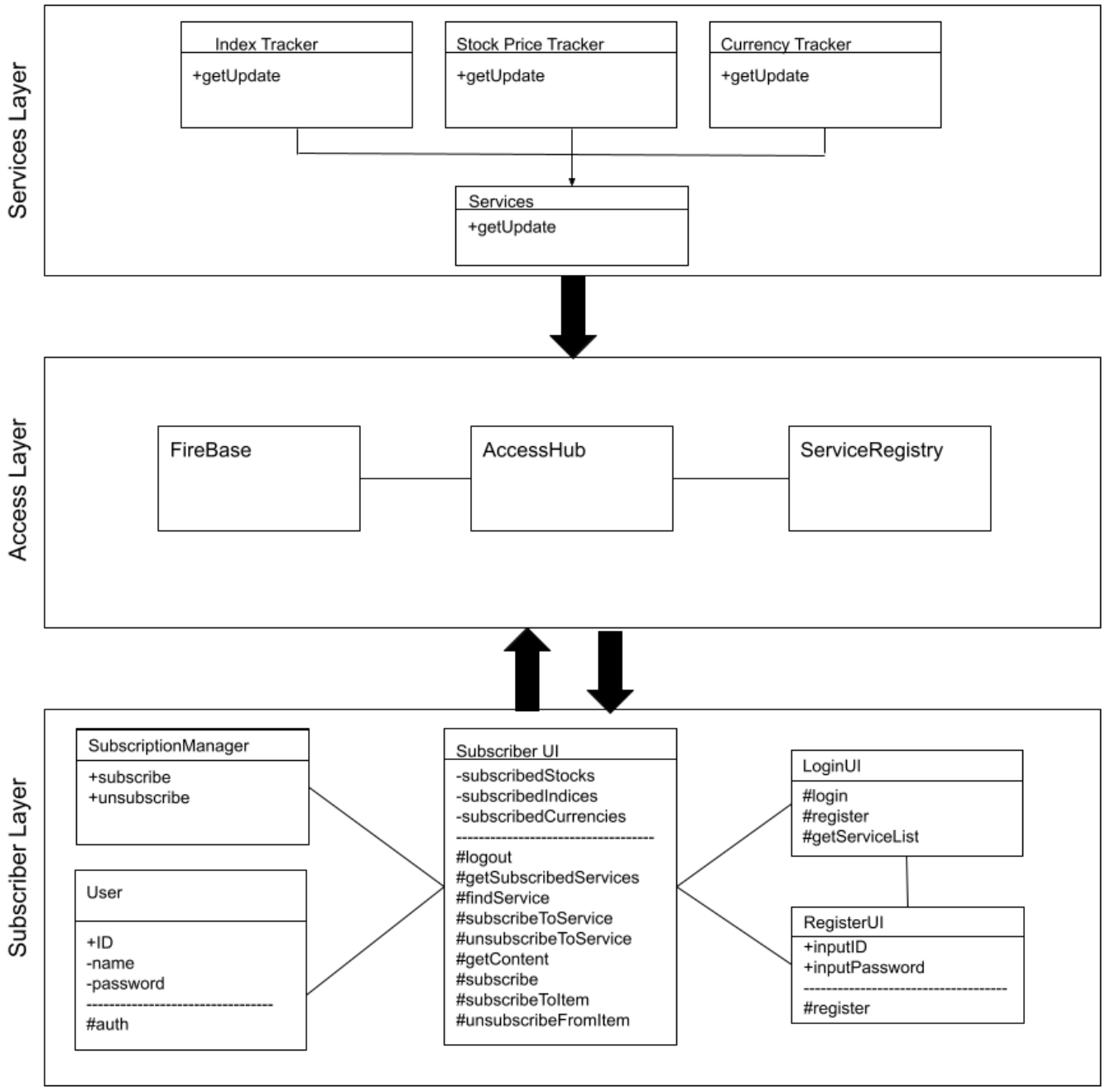
2.6: Directory

2.6.1: Glossary

Stock	A security that represents the ownership of a fraction of a corporation
Index	A method to track the performance of a group of stocks in a standardized way
Service-Oriented Architecture	A way to make software components reusable via service interfaces
Publish-Subscribe Pattern	An architectural design pattern that provides a framework for exchanging messages between publishers and subscribers

3. Module View

3.1: Primary Presentation



3.2: Element Catalogue
3.2.1: Elements and Properties

Element #1: Services Layer

Responsibilities
- Holds services and sends updates to Access Layer

Visibility of interfaces
- Private

Implementation information
- Mapping to source code units
  - /services
- Test information
  - Tested independently of other layers iteratively and with varying numbers of subscribers and services
- Management information
  - First phase of development completed by March 15th, 2021
  - Second phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Developed to be compatible with IBM Cloud Services and Node.js
- Revision history
  - Version 1 under development as of March 5th, 2021
  - Version 2 under development as of March 15th, 2021
  - Version 2 under review as of March 25th, 2021

Element #2: Index Tracker

Responsibilities
- Tracks stock indices and sends information to Services module

Visibility of interfaces
- Protected

Implementation information
- Mapping to source code units
  - /services/indextracker.js
- Test information
  - Tested independently of other layers iteratively and with varying numbers of subscribers and services
- Management information
  - First phase of development completed by March 15th, 2021
  - Second phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Developed to be compatible with twelvedata API
- Revision history
  - Version 1 under development as of March 5th, 2021

- Version 2 under development as of March 15th, 2021
- Version 2 under review as of March 25th, 2021

Element #3: Stock Price Tracker

Responsibilities
- Tracks stock prices and sends information to Services module

Visibility of interfaces
- Protected

Implementation information
- Mapping to source code units
  - /services/stocktracker.js
- Test information
  - Tested independently of other layers iteratively and with varying numbers of subscribers and services
- Management information
  - First phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Developed to be compatible with twelvedata API
- Revision history
  - Version 1 under development as of March 25th, 2021
  - Version 1 under review as of April 5th, 2021

Element #4: Currency Tracker

Responsibilities
- Tracks currency values and sends information to Services module

Visibility of interfaces
- Protected

Implementation information
- Mapping to source code units
  - /services/currencytracker.js
- Test information
  - Tested independently of other layers iteratively and with varying numbers of subscribers and services
- Management information
  - First phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Developed to be compatible with twelvedata API
- Revision history
  - Version 1 under development as of March 25th, 2021
  - Version 1 under review as of April 5th, 2021

Element #5: Services

Responsibilities

- Retrieves information from all services in Services Layer

Visibility of interfaces

- Public

Implementation information

- Mapping to source code units
  - /src/stocks.js
  - /src/indices.js
  - /src/currencies.js
- Test information
  - Tested iteratively and variably with existing services in Services Layer
- Management information
  - First phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Will be developed based on individual services in Services Layer
- Revision history
  - Version 1 under development as of March 5th, 2021
  - Version 2 under development as of March 15th, 2021
  - Version 2 under review as of March 25th, 2021

Element #6: Access Layer

Responsibilities

- Houses stock data and service registry
- Forwards information from Services Layer to relevant subscribers in Subscriber Layer
- Receives information from Subscriber Layer to maintain Service Registry

Visibility of interfaces

- Public

Implementation information

- Mapping to source code units
  - /src/firebase.js
  - /src/firebase-user.js
  - /src/screens/login.js
  - /src/screens/dashboard.js
- Test information
  - Tested iteratively and variably with existing services in Services Layer and for varying numbers of subscribers in Subscriber Layer
- Management information
  - First phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Developed to be compatible with FireBase
- Revision history

- Version 1 under development as of March 15th, 2021
- Version 2 under development as of March 25th, 2021
- Version 2 under review as of April 5th, 2021

Element #7: FireBase

Responsibilities
- Holds stock data

Visibility of interfaces
- Protected

Implementation information
- Mapping to source code units
  - /src/firebase.js
- Test information
  - Tested iteratively and variably with existing services in Services Layer and for varying numbers of subscribers in Subscriber Layer
- Management information
  - First phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints
  - Developed using Google Services (FireBase)
- Revision history
  - Version 1 under development as of March 15th, 2021
  - Version 2 under development as of March 25th, 2021
  - Version 2 under review as of April 5th, 2021

Element #8: AccessHub

Responsibilities
- Acts as a communications bus between system layers

Visibility of interfaces
- Public

Implementation information
- Mapping to source code units
  - src/firebase.js
  - src/firebase-user.js
  - src/dashboard.js
- Test information
  - Tested iteratively and variably with existing services in Services Layer and for varying numbers of subscribers in Subscriber Layer and with varying amounts of data to be transmitted
- Management information
  - First phase of development completed by March 25th, 2021
  - Final phase of development completed by April 5th, 2021
- Implementation constraints

    - Developed based on twelvedata API and Google FireBase
- Revision history
    - Version 1 under development as of March 15th, 2021
    - Version 1 under review as of March 25th, 2021

Element #9: ServiceRegistry

Responsibilities
- Maintains a collection of subscriptions for each user

Visibility of interfaces
- Public

Implementation information
- Mapping to source code units
    - /src/firebase-user.js
- Test information
    - Tested iteratively and variably with existing services in Services Layer and for varying numbers of subscriptions
- Management information
    - First phase of development completed by March 25th, 2021
    - Final phase of development completed by April 5th, 2021
- Implementation constraints
    - Developed using Google FireBase
- Revision history
    - Version 1 under development as of March 15th, 2021
    - Version 2 under development as of March 25th, 2021
    - Version 2 under review as of April 5th, 2021

Element #10: Subscriber Layer

Responsibilities
- Provides a user interface and management system for existing and new subscribers
- Sends subscriber information to Access Layer
- Receives service information from Services Layer

Visibility of interfaces
- Public

Implementation information
- Mapping to source code units
    - /src/screens/signup.js
    - /src/screens/login.js
    - /src/screens/dashboard.js
- Test information
    - Tested iteratively and variably with existing services in Services Layer and for varying numbers of subscriptions and unsubscriptions
- Management information
    - First phase of development completed by March 25th, 2021

    - Final phase of development completed by April 5th, 2021
- Implementation constraints
    - Developed using Node.js and React frameworks
- Revision history
    - Version 1 under development as of March 15th, 2021
    - Version 2 under development as of March 25th, 2021
    - Version 2 under review as of April 5th, 2021

Element #11: SubscriptionManager

Responsibilities
- Manages user-controlled activities such as subscribing or unsubscribing from a service

Visibility of interfaces
- Protected

Implementation information
- Mapping to source code units
    - /src/Signup.js
    - /src/Login.js
- Test information
    - Tested iteratively and variably with existing services in Services Layer and for varying numbers of subscriptions and unsubscriptions
- Management information
    - First phase of development completed by March 25th, 2021
    - Final phase of development completed by April 5th, 2021
- Implementation constraints
    - Developed using back-end Node.js with React JS framework
- Revision history
    - Version 1 under development as of March 15th, 2021
    - Version 2 under development as of March 25th, 2021
    - Version 2 under review as of April 5th, 2021

Element #12: User

Responsibilities
- Acts as a representation of users that have saved a name and password to use with the application

Visibility of interfaces
- Protected

Implementation information
- Mapping to source code units
    - /src/UpdateProfile.js
- Test information
    - Tested iteratively for varying numbers of subscriptions and unsubscriptions and varying numbers and types of service updates
- Management information

        - First phase of development completed by March 15th, 2021
        - Second phase of development completed by March 25th, 2021
        - Final phase of development completed by April 5th, 2021
- Implementation constraints
    - Developed using Node.js with React JS
- Revision history
    - Version 1 under development as of March 15th, 2021
    - Version 2 under development as of March 25th, 2021
    - Version 2 under review as of April 5th, 2021

Element #13: SubscriberUI

Responsibilities

- Provides a graphical interface for users to view and modify their subscriptions

Visibility of interfaces

- Public

Implementation information

- Mapping to source code units
    - /src/screens/dashboard.js
- Test information
    - Tested iteratively and independently for subscribing and unsubscribing to services
- Management information
    - First phase of development completed by March 25th, 2021
    - Final phase of development completed by April 5th, 2021
- Implementation constraints
    - Developed using Node.js with React JS
- Revision history
    - Version 1 under development as of March 25th, 2021
    - Version 1 under review as of April 5th, 2021

Element #14: LoginUI

Responsibilities

- Provides a graphical interface for users to login to their personal subscriptions dashboard

Visibility of interfaces

- Public

Implementation information

- Mapping to source code units
    - /src/screens/login.js
- Test information
    - Tested iteratively and independently for subscribing and unsubscribing to services
- Management information
    - First phase of development completed by March 25th, 2021

        - Final phase of development completed by April 5th, 2021
    - Implementation constraints
        - Developed using Node.js with React JS
    - Revision history
        - Version 1 under development as of March 25th, 2021
        - Version 1 under review as of April 5th, 2021

Element #15: Register UI

Responsibilities

- Provides a graphical interface for users to register a name and password in order to access the application's services

Visibility of interfaces

- Public

Implementation information

- Mapping to source code units
    - /src/signup.js
- Test information
    - Tested iteratively and independently for with varying name and password combinations
- Management information
    - First phase of development completed by March 25th, 2021
    - Final phase of development completed by April 5th, 2021
- Implementation constraints
    - Developed using Node.js with React JS
- Revision history
    - Version 1 under development as of March 25th, 2021
    - Version 1 under review as of April 5th, 2021

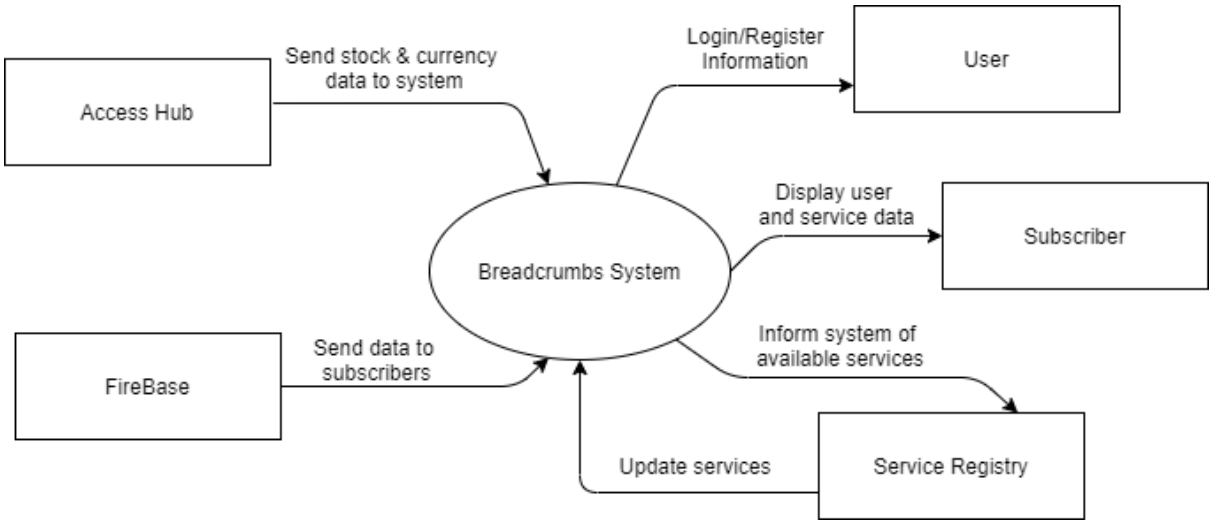
3.2.2 Relationships

Associations and dependencies as per Primary Presentation (see section 3.1).

- Access Layer *uses* Services Layer
- Subscriber Layer *uses* Access Layer
- Access Layer *uses* Subscriber Layer

- Index Tracker *inherits* Services
- Stock Price Tracker *inherits* Services
- Currency Tracker *inherits* Services

3.3: Context Diagram



3.4: Variability Guide

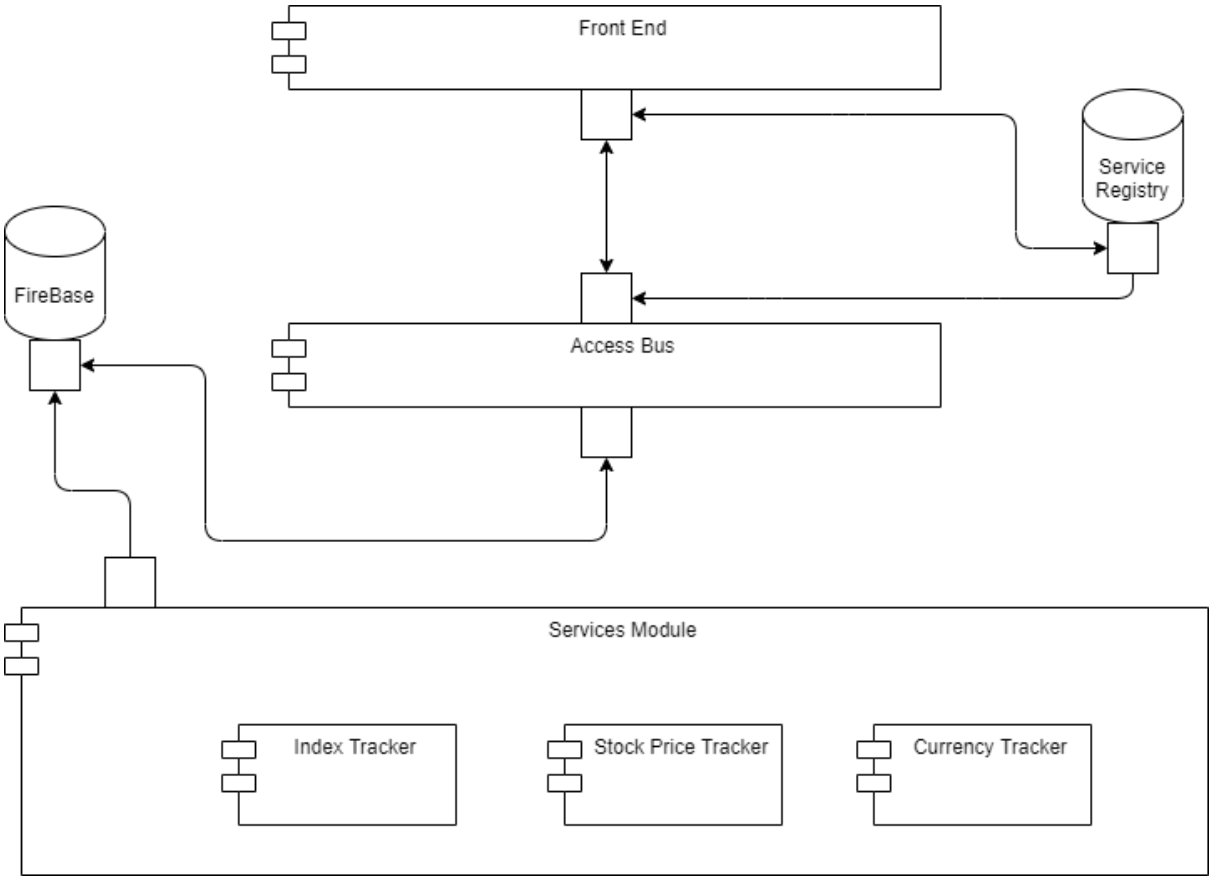
- The available services in ServiceRegistry can change as services are added and removed
- Services can be added to or removed from the Services Layer, with all services inheriting Service
- The information displayed in SubscriberUI can change depending on what services the user is subscribed to
- Data in FireBase changes as each service sends it new information

3.5: Rationale

A layered architectural pattern allows for each module to be interconnected while allowing them to not be dependent on each other. Thus reducing coupling. This pattern also has a low implementation complexity compared to other patterns and is suitable for our system which is of small to moderate size.

4. Component-and-Connector View

4.1: Primary Presentation



4.2: Element Catalogue

4.2.1: Elements and Properties

Component #1: Index Tracker

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements.

Functionality

- Retrieve stock data from an external dataset API, aggregate these values into indexes, calculate value of each index based on the values of its stocks, and send this data to FireBase to be stored. Afterwards, send an update to the Access Bus to inform it that the index data in FireBase has changed.

Security

- This component has no security features

Concurrency

- This component executes concurrently with the other trackers in the Services Module

Modifiability

- This component has the ability to be modified to support service changes such as extra subscription options or index predictions

Tier

- Services Module

Component #2: Stock Price Tracker

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements.

Functionality

- Retrieve stock data from an external dataset or API, and send this data to FireBase to be stored. Afterwards, send an update to the Access Bus to inform it that the stock data in FireBase has changed.

Security

- This component has no security features

Concurrency

- This component executes concurrently with the other trackers in the Services Module

Modifiability
- This component has the ability to be modified to support service changes such as extra subscription options or stock predictions

Tier
- Services Module

Component #3: Currency Tracker

Reliability
- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance
- The response time of this component will be quick

Resource Requirements
- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements.

Functionality
- Retrieve currency data from an external dataset or API, and send this data to FireBase to be stored. Afterwards, send an update notification to the Notification Bus to inform it that the stock data in FireBase has changed.

Security
- This component has no security features

Concurrency
- This component executes concurrently with the other trackers in the Services Module

Modifiability
- This component has the ability to be modified to support service changes such as extra subscription options or currency predictions

Tier
- Services Module

Component #4: Services Module

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. Its storage requirement would be the sum of the components contained within

Functionality

- Send data from the components within to FireBase, send updates from within to the Access Bus, and send updates to the Service Registry regarding the availability of the components within.

Security

- This component has no security features

Concurrency

- This component executes concurrently with other components.

Modifiability

- This component has the ability to be modified to support the addition and removal of the services it contains.

Tier

- Services Module

Component #5: Access Bus

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance

- The response time of this component will be generally fast, however response time may vary depending on the client's connection speed.

Resource Requirements

- This component will require enough storage to manage a list of subscribers for each service.

Functionality

- Act as a common communications point for clients and servers

Security

- User data is encrypted.

Concurrency

- This component executes concurrently with other components

Modifiability

- This component has the ability to be modified to support the addition and removal of services.

Tier

- Access Module

Component #7: FireBase

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the Google Firebase Service.

Performance

- The response time of this component will be fast.

Resource Requirements

- This component has a high storage requirement, as it needs to store all data sent to it from the Services Module, as well as extra space to support the creation of additional services.

Functionality

- Stores data retrieved in the Services Module

Concurrency

- This component is a database that exists independently of the rest of the system

Modifiability

- This component has the ability to be modified to support the addition and removal of services.

Tier

- Database Module

Component #8: Service Registry

Reliability
- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance
- The response time of this component will be fast.

Resource Requirements
- This component has a high storage requirement, as it needs to store information about all available services and support the addition of more services.

Functionality
- Stores information about all services and their availability.

Concurrency
- This component is a database that exists independently of the rest of the system

Modifiability
- This component has the ability to be modified to support the addition and removal of services.

Tier
- Database Module

Component #9: Front End

Reliability
- This component will fail if the back end components fail, as it won't be able to access account data.

Performance
- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component needs to be able to manage large amounts of data retrieved from FireBase

Functionality

- Display all available services to the user, and allow them to subscribe to specific services. Displays data from all subscribed services to the user.

Concurrency

- Users can receive notifications from multiple subscribed channels concurrently

Modifiability

- This component has the ability to be modified to support the addition and removal of services.

Tier

- Front End Module

4.2.2: Relationships and Properties

Associations and dependencies as per Primary Presentation (see section 3.1).

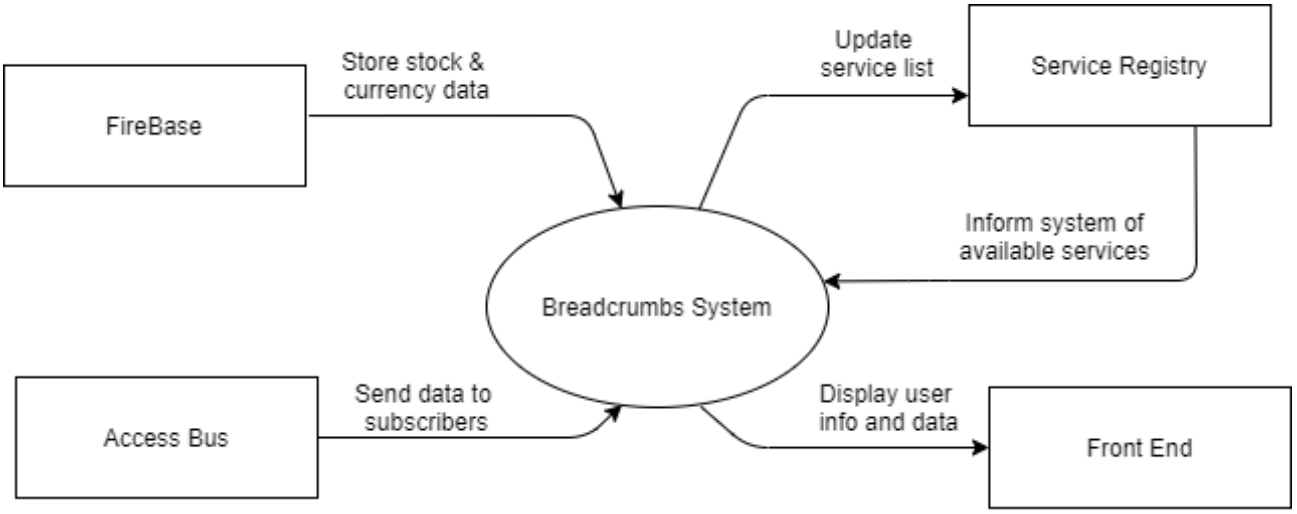
- Front End *uses* Access Bus
- Access Bus *uses* Front End

- Front End *uses* Service Registry
- Service Registry *uses* Front End

- Services Modules *publishes to* FireBase

- Index Tracker *is-a* Service Module
- Stock Price Tracker *is-a* Service Module
- Currency Tracker *is-a* Service Module

4.3: Context Diagram



4.4: Variability Guide

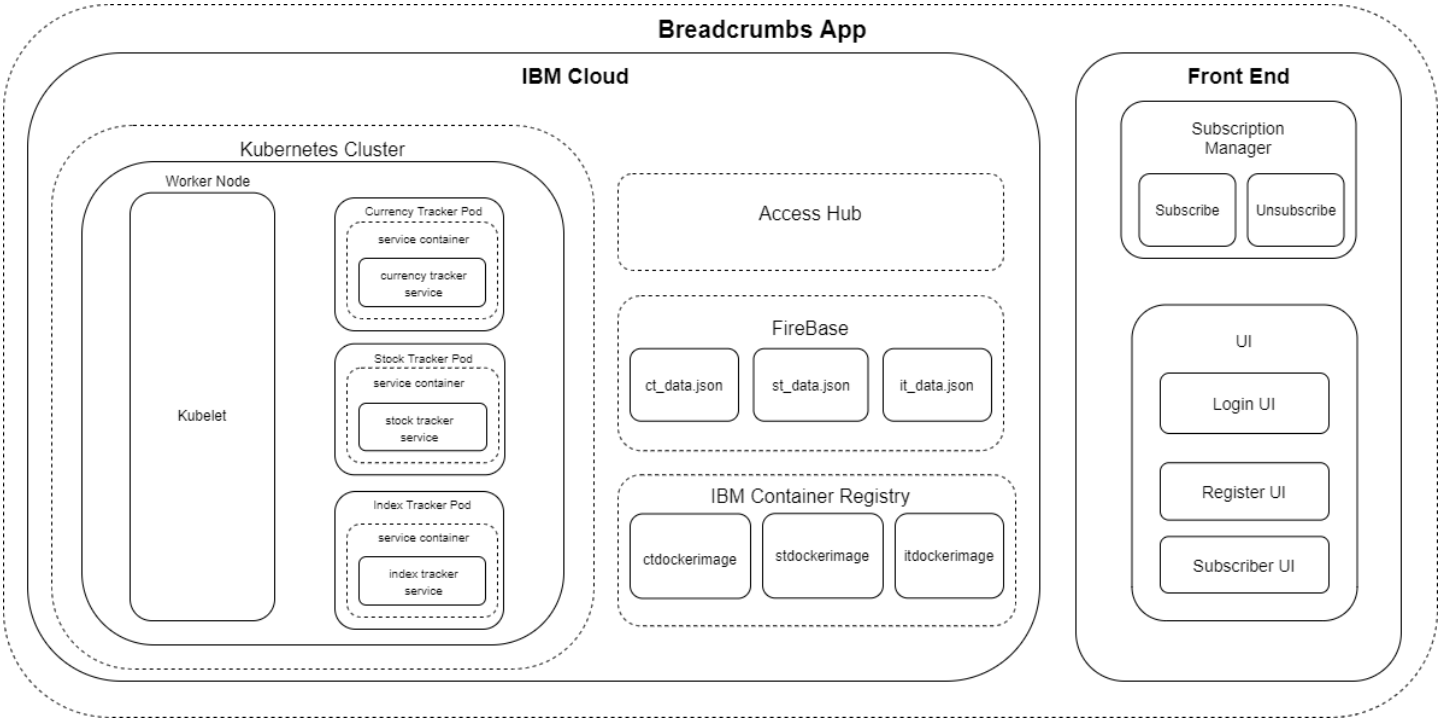
- The available services in the Service Registry can change as services are added and removed
- Services can be added to or removed from the Services Module
- The information displayed in the Front End can change depending on what services the user is subscribed to
- Data in FireBase changes as each service sends it new information

4.5: Rationale

The component-and-connector view was used as it provides a comprehensive overview of the abstract software components in the system using general but descriptive terminology which can be understood by a wide variety of software developers.

5. Allocation View

5.1: Primary Presentation



5.2: Element Catalogue

5.2.1: Elements and Properties

Element #1: IBM Cloud

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality

- IBM cloud serves as the back end infrastructure of the project

Concurrency

- This component executes concurrently with other components

Modifiability

- This component is not modifiable in the scope of this project

Tier

- Back End Module

Element #2: Kubernetes Cluster

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality

- Kubernetes clusters of compute hosts are deployed and manage containerized apps on IBM cloud

Concurrency

- This component executes concurrently with other components

Modifiability

- This component is not modifiable in the scope of this project

Tier

- Back End Module

Element #3: Worker Node

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. Worker nodes are instantiated on IBM Cloud

Functionality
- Encapsulate runtime execution on IBM Cloud

Concurrency
- This component executes concurrently with other components

Modifiability
- This component is not modifiable in the scope of this project

Tier
- Back End Module

Element #4: Kubelet

Reliability
- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service

Performance
- The response time of this component will be quick

Resource Requirements
- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality
- Abstract management of Kubernetes clusters

Concurrency
- This component executes concurrently with other components

Modifiability
- This component is not modifiable in the scope of this project

Tier
- Back End Module

Element #5: Currency Tracker Pod

Reliability

- This component is unlikely to fail. Services are highly cohesive and minimally coupled via Dockerization

Performance
- The response time of this component will be quick

Resource Requirements
- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality
- Retrieve currency data from an external dataset or API, and send this data to subscribers

Concurrency
- This component executes concurrently with other components

Modifiability
- This component has the ability to be modified to support service changes such as extra subscription options or currency predictions

Tier
- Service Module

Element #6: Stock Tracker Pod

Reliability
- This component is unlikely to fail. Services are highly cohesive and minimally coupled via Dockerization

Performance
- The response time of this component will be quick

Resource Requirements
- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality
- Retrieve stock data from an external dataset or API, and send this data to subscribers

Concurrency
- This component executes concurrently with other components

Modifiability

- This component has the ability to be modified to support service changes such as extra subscription options or stock predictions

Tier

- Service Module

Element #7: Index Tracker Pod

Reliability

- This component is unlikely to fail. Services are highly cohesive and minimally coupled via Dockerization

Performance

- The response time of this component will be quick

Resource Requirements

- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality

- Retrieve index data from an external dataset or API, and send this data to subscribers

Concurrency

- This component executes concurrently with other components

Modifiability

- This component has the ability to be modified to support service changes such as extra subscription options or index predictions

Tier

- Service Module

Element #9: FireBase

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance

- The response time of this component will be fast.

Resource Requirements

- This component has a high storage requirement, as it needs to store all data sent to it from the Worker Node, as well as extra space to support the creation of additional services.

Functionality

- Stores data retrieved in the Services Module

Concurrency

- This component is a database that exists independently of the rest of the system

Modifiability

- This component has the ability to be modified to support the addition and removal of services.

Tier

- Database Module

Element #10: IBM Container Registry

Reliability

- This component is unlikely to fail. Failure of this component would require failure of the IBM Cloud service.

Performance

- The response time of this component will be fast.

Resource Requirements

- This component has a high storage requirement, as it needs to store information about all available services and support the addition of more services.

Functionality

- Stores information about all services and their availability.

Concurrency

- This component is a database that exists independently of the rest of the system

Modifiability

- This component has the ability to be modified to support the addition and removal of services.

Tier

- Database Module

Element #11: Front End

Reliability

- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance

- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component needs to be able to manage large amounts of data retrieved from FireBase

Functionality

- This component houses all components regarding UI and connection to services.

Concurrency

- This component exists independently from IBM Cloud components, and so is able to run concurrently with it.

Modifiability

- This component has the ability to be modified to support the addition and removal of services.

Tier

- Front End Module

Element #12: Subscription Manager

Reliability

- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance

- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component does not store any information locally, so has minimal space requirements

Functionality

- Contains all components that handle subscribing to and unsubscribing from services

Concurrency

- This component can run concurrently with other front end modules.

Modifiability

- This component has the ability to be modified to support new subscription options.

Tier

- Front End Module

Element #13: Subscribe

Reliability

- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance

- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component does not store any information locally, so has minimal space requirements

Functionality

- Subscribes the user to services.

Concurrency

- This component can run concurrently with other front end modules.

Modifiability

- This component has the ability to be modified to support new subscription options.

Tier

- Front End Module

Element #14: Unsubscribe

Reliability

- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance

- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component does not store any information locally, so has minimal space requirements

Functionality

- Unsubscribes the user from services.

Concurrency

- This component can run concurrently with other front end modules.

Modifiability

- This component has the ability to be modified to support new subscription options.

Tier

- Front End Module

Element #15: UI

Reliability

- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance

- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component must be able to store all UI components, and accommodate new ones.

Functionality

- Contains all UI components

Concurrency

- This component can run concurrently with other front end modules.

Modifiability

- This component has the ability to be modified to support new UI components.

Tier

- Front End Module

Element #16: Login UI

Reliability

- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance

- The response time of this component will be generally fast, however latency when connecting to back end services may vary depending on the user's internet speed.

Resource Requirements

- This component has minimal resource requirements

Functionality

- Allows the user to log in, allowing them access to subscription. If the user does not have an account, they can access Register UI from this component.

Concurrency

- This component can run concurrently with other front end modules.

Modifiability

- This component has the ability to be modified to support new UI components.

Tier
- Front End Module

Element #17: Register UI

Reliability
- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance
- The response time of this component will be quick

Resource Requirements
- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality
- The interface that lets the user input ID and password.

Concurrency
- This component can run concurrently with other front end modules.

Modifiability
- This component has the ability to be modified to support new UI components.

Tier
- Front End Module

Element #18: Subscriber UI

Reliability
- This component is unlikely to fail since docker build images are containerized before deployment/during continuous integration

Performance
- The response time of this component will be quick

Resource Requirements
- This component will require very few resources. All data is stored externally, eliminating the need for large storage requirements

Functionality

- The interface that lets the user subscribe to stocks, currencies, or indices.

Security

- Subscriber information if hidden from other modules

Concurrency

- This component can run concurrently with other front end modules.

Modifiability

- This component has the ability to be modified to support new UI components.

Tier

- Front End Module

5.2.2: Relationships and Properties

- Application *contains* IBM Cloud (instance of)
- Application *contains* Front End

- IBM Cloud *contains* Kubernetes Cluster
- IBM Cloud *contains* Google FireBase
- IBM Cloud *contains* IBM Container Registry

- Kubernetes Cluster *contains* Worker Node
- Worker Node *contains* Kubelet

- Worker Node *contains* Currency Tracker Pod
- Worker Node *contains* Stock Tracker Pod
- Worker Node *contains* Index Tracker Pod

- Currency Tracker Pod *contains* Service Container (instance of)
- Stock Tracker Pod *contains* Service Container (instance of)
- Index Tracker Pod *contains* Service Container (instance of)

- Service Container *contains* Currency Tracker Service
- Service Container *contains* Stock Tracker Service
- Service Container *contains* Index Tracker Service

- IBM Event Stream *contains* Notification Bus
- Google FireBase *contains* Currency Tracker Data
- Google FireBase *contains* Stock Tracker Data
- Google FireBase *contains* Index Tracker Data

- IBM Container Registry *contains* Currency Tracker Docker Image

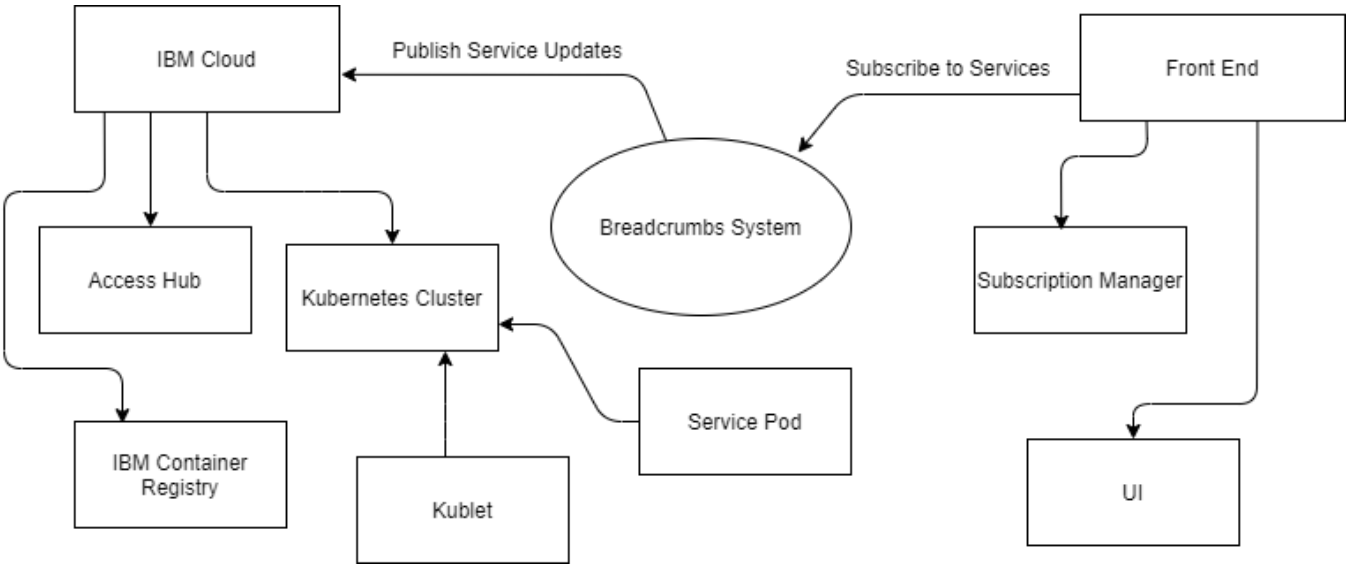
- IBM Container Registry *contains* Stock Tracker Docker Image
- IBM Container Registry *contains* IndexTracker Docker Image

- Front End *contains* Subscription Manager
- Front End *contains* UI

- Subscription Manager *contains* Subscribe Function
- Subscription Manager *contains* Unsubscribe Function

- UI *contains* Login UI
- UI *contains* Register UI
- UI *contains* Subscriber UI

5.3: Context Diagram



5.4: Variability Guide

- The available services in the Container Registry can change as services are added and removed
- New containers can be added to or removed from the Worker Node to accommodate changes to services
- New nodes can be deployed in the Kubernetes Cluster to scale with large numbers of services
- The information displayed in SubscriberUI can change depending on what services the user is subscribed to

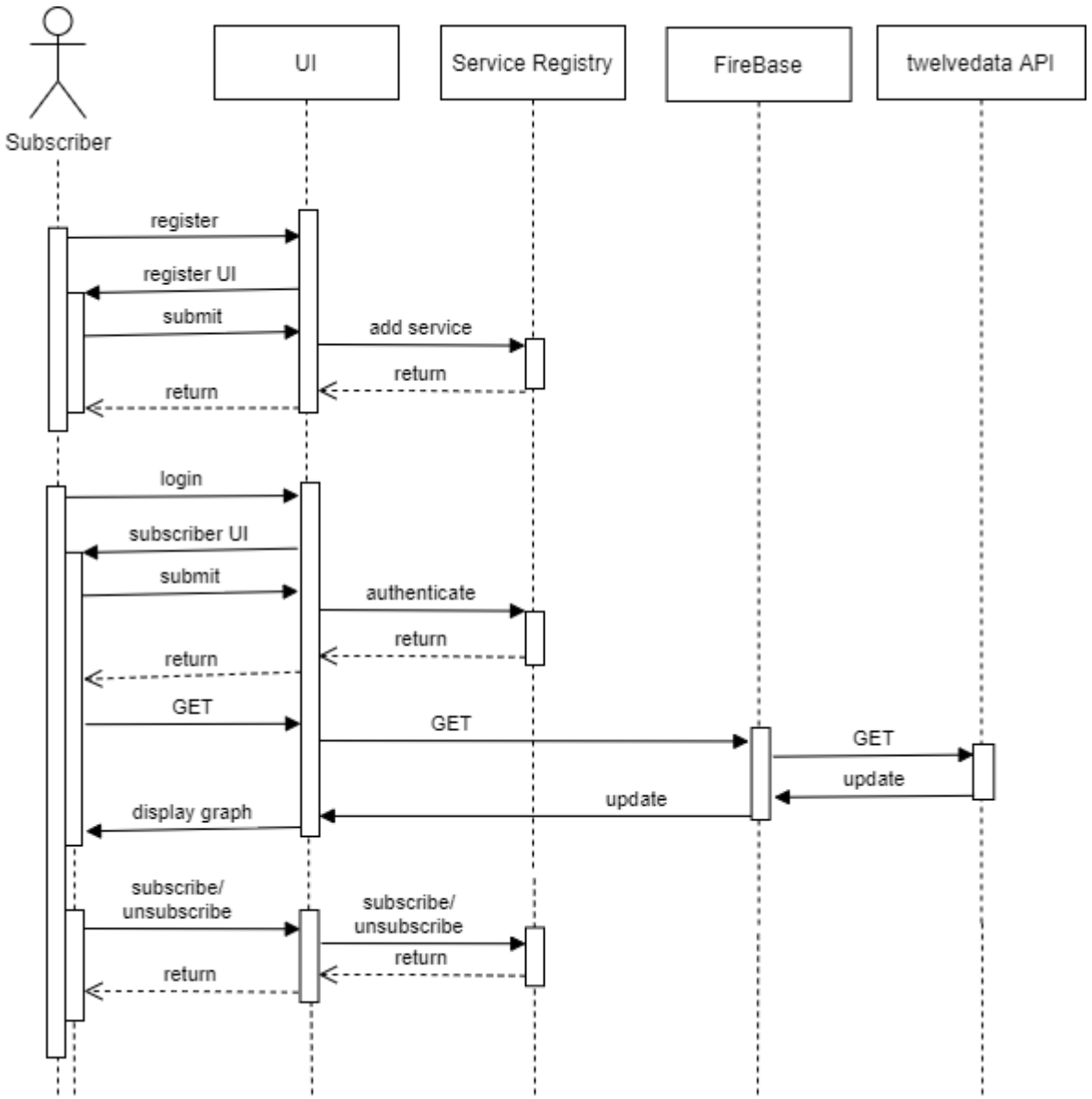
- Data in FireBase changes as each service sends it new information

5.5: Rationale

The deployment view describes the environment in which the system is executed. The view provides information about how the system will exist in the real world using existing professional software resources. The view also allows for analysis of performance, availability, and modifiability.

6. Sequence View

6.1: Primary Presentation



6.2: Element Catalogue
6.2.1: Elements and Properties

Element #1: Subscriber

Binding Time
- Runtime

Triggers
- Agent of free will

Return Values
- N/A

Scope
- Front end

Element #2: UI

Binding Time
- Compile time

Triggers
- User actions including registration, login, subscription/unsubscription, and request for updates from services

Return Values
- Register UI, Subscriber UI, display service

Scope
- Front end

Element #3: Service Registry

Binding Time
- Compile time

Triggers
- Signal for new subscriber registered in the system from UI or existing subscriber removed from system (deleted account)

Return Values
- New service added, subscriber authenticated, subscribed/unsubscribed

Scope
- Front end, Back end

Element #4: FireBase

Binding Time
- Compile time

Triggers
- Received request for updated data propagated from user

Return Values
- Service updates

Scope
- Back end

Element #5: twelvedata API

Binding Time
- Compile time

Triggers
- Received request for updated data propagated from user

Return Values
- Data updates

Scope
- Back end

6.2.2: Relationships and Properties

- Subscriber *executes* register
- Subscriber *executes* login
- Subscriber *executes* GET
- Subscriber *executes* subscriber/unsubscribe
- Subscriber *executes* submit

- UI *returns* register_UI
- UI *returns* subscriber_UI
- UI *returns* display_graph
- UI *returns* return

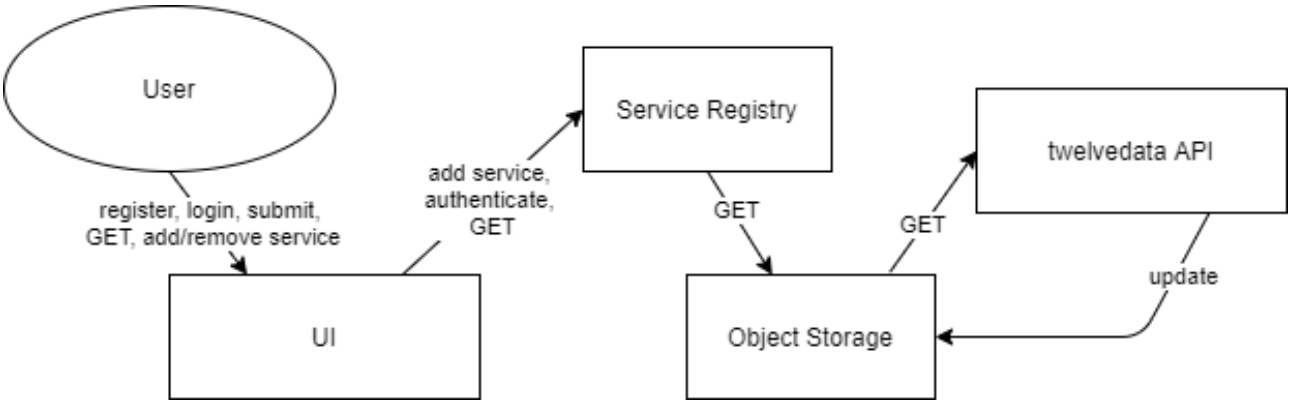
- UI *executes* add_service
- UI *executes* authenticate
- UI *executes* GET

- Service Registry *returns* return

- FireBase *executes* GET
- FireBase *returns* update

- twelvedata API *executes* update

6.3: Context Diagram



6.4: Variability Guide

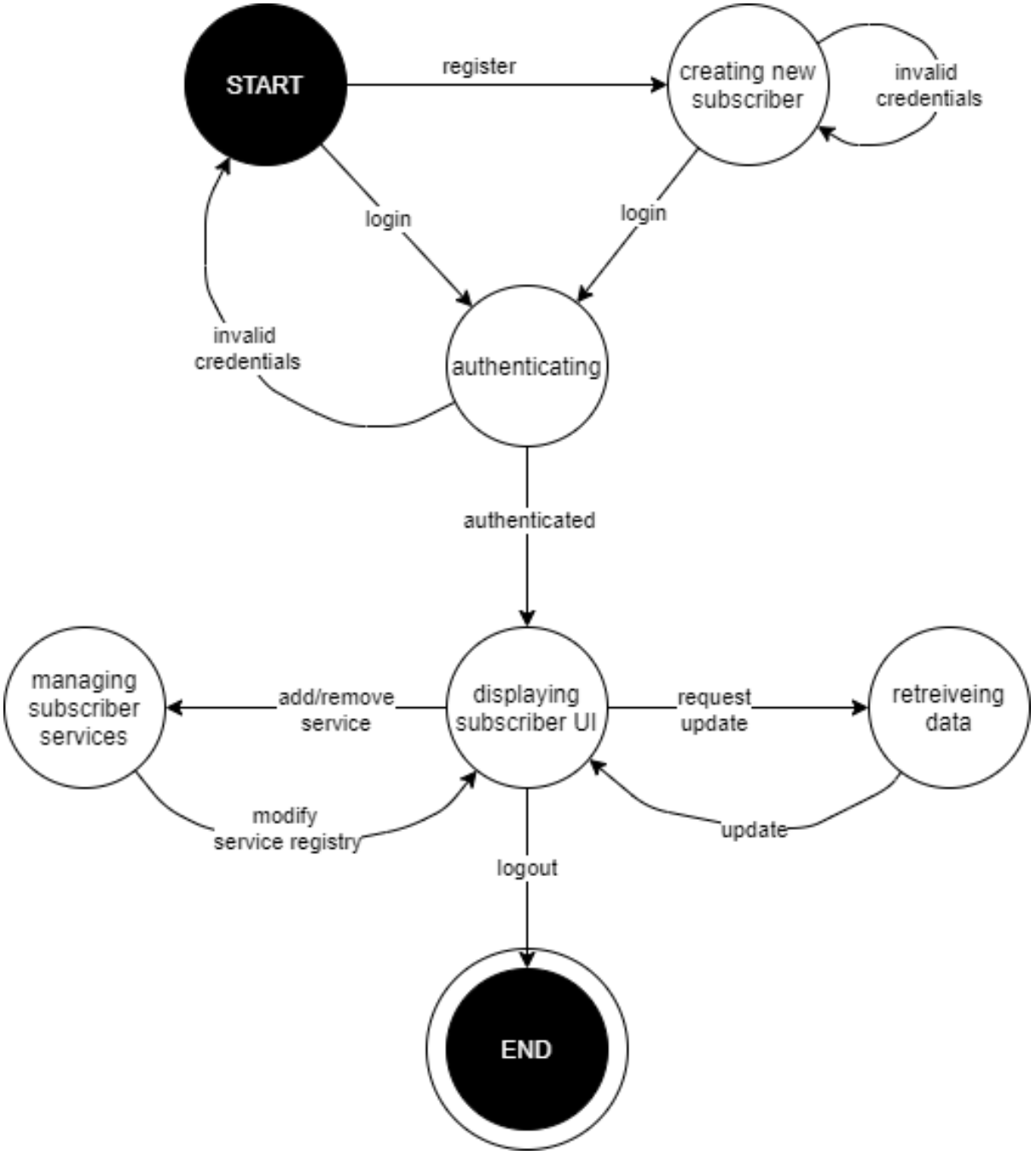
- UI may be accessed from a browser on desktop or mobile
- Service Registry elements may change as new subscriber are added and removed
- FireBase elements may change according to data updates
- twelvedata API may require calls for stock data, index data, or currency data

6.5: Rationale

The sequence diagram was used to describe the system over a period of time during which the user will be interacting with various elements of the system. This view may inform concerns about error detection and correction, resulting in better prediction of possible faults and failures in the system during runtime.

7. User Interface State View

7.1: Primary Presentation



7.2: Element Catalogue
7.2.1: Elements and Properties

State #1: Creating new subscriber

Binding Time
- Runtime

Triggers
- User action “register”

State changes
- No state change if user enters invalid credentials
- Go to “authenticating” state if valid credentials

Scope
- Front end

State #2: Authenticating

Binding Time
- Runtime

Triggers
- User action “login”

State changes
- Go to start state if user enters invalid credentials
- Go to “displaying subscriber UI” if valid credentials

Scope
- Front end

State #3: Displaying subscriber UI

Binding Time
- Runtime

Triggers
- System action “authenticated” (after user login or register)
- System action “modify service registry”
- System action “update”

State changes
- Go to “managing subscriber services” if user action “add/remove service”
- Go to “retrieving data” if user action ”request update”
- Go to end state if user action “logout”

Scope
- Front end

State #4: Managing subscriber services

Binding Time
- Runtime

Triggers
- User action “add/remove service”

State changes
- Go to “displaying subscriber UI” if system action “modify service registry”

Scope
- Front end

State #5: Retrieving data

Binding Time
- Runtime

Triggers
- User action “request update”

State changes
- Go to “displaying subscriber UI” if system action “update”

Scope
- Back end

7.2.2: Relationships and Properties

- Start *goes-to* creating_new_subscriber
- Start *goes-to* authenticating

- Creating_new_subscriber *goes-to* creating_new_subscriber
- Creating_new_subscriber *goes-to* authenticating

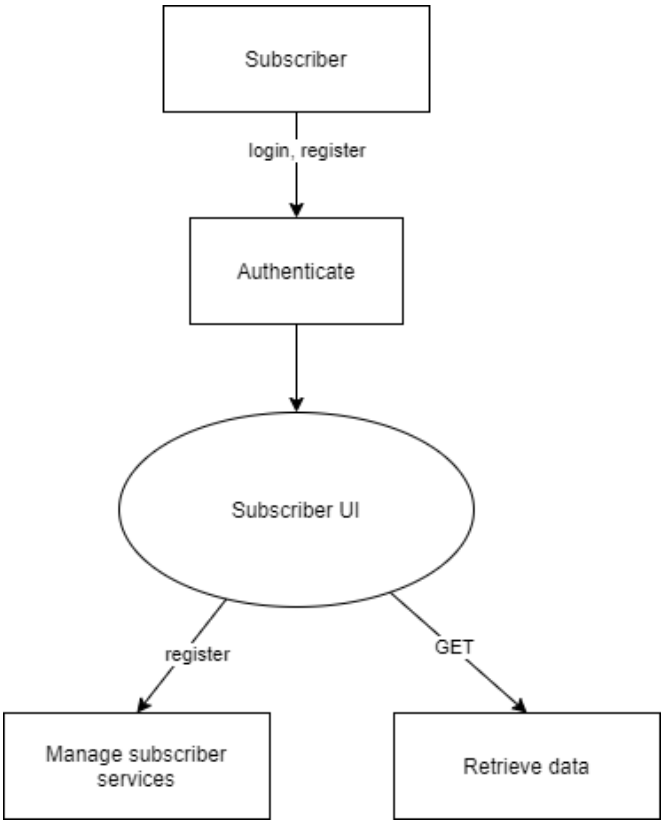
- Authenticating *goes-to* start
- Authenticating *goes-to* displaying_subscriber_UI

- Displaying_subscriber_UI *goes-to* managing_subscriber_services

- Displaying_subscriber_UI *goes-to* retrieving_data
- Displaying_subscriber_UI *goes-to* end

- Managing_subscriber_services *goes-to* displaying_subscriber_UI
- Retrieving_data *goes-to* displaying_subscriber_UI

7.3: Context Diagram



7.4: Variability Guide

- Management of subscriber services may vary depending on which available service is added or removed
- Data retrieval may vary depending on which service the data is being requested for

7.5: Rationale

The state diagram was used to model the system to clarify which actions are performed by the user as well as which actions are performed by the system, and in which order. This view focuses on the front end of the application, allowing for further accuracy of predictions about the system’s potential faults and failures as experienced by the user, and how they should be handled.