

University of Houston-Clear Lake

College of Science and Engineering

Project Report



Advanced Operating Systems

Currency Trading System [Indirect Communication]

Umar Abdul Aziz
2310655

TABLE OF CONTENTS

I.	INTRODUCTION	3
II.	DATA STRUCTURE AND ALGORITHMS	6
III.	PROBLEM SOLVING – DEALING ROOM SERVER	9
IV.	PROBLEM SOLVING – CLIENT	12
V.	DIAGRAMATIC REPRESENTATION OF THE SYSTEM	15
VI.	SCREENSHOTS	18
VII.	CONCLUSION	23

INTRODUCTION

The world of cryptocurrency trading is dynamic and time-sensitive; it often requires real-time information and super-efficient forms of communication between entities to perform informed and effective decision-making for success. This project implements, using Java, a Publish-Subscribe Cryptocurrency Dealing Room System that aims to solve the problems of distributed communication in trading systems. The main purpose of the following project is to utilize a few of Java's advanced APIs for distributed systems to build a system that can simulate indirect communication across a dealing room. It will also enable multiple users to publish and subscribe to updates in cryptocurrency news in real time. This methodology will give one a practical example not only of the Publish-Subscribe model of communication but also of how a group messaging API in Java can be used in designing a distributed, event-driven system.

The Publish-Subscribe model, further shortened as pub-sub, is an indirect communication pattern that allows entities to communicate with each other without direct interaction. In this model, publishers publish information to interested parties-subscribers-by broadcasting messages to a central system, or "bulletin board." A subscriber subscribes by declaring his interest in specific topics like any updates related to Bitcoin, Ethereum, or generally other cryptocurrencies. The system automatically forwards notifications in case of new information on a subscribed topic. This further mocks the real-world dealing room environment where updates are received from various external sources, and subscribers or dealers receive the latest data on stocks or assets of interest in an asynchronous manner.

Project Objectives

The goal of this project is to implement, in a distributed fashion, a simple cryptocurrency dealing room using Java's distributed API, with a main focus on the Publish-Subscribe model for indirect communications. Users of this kind of a dealing room should be able to:

1. **CryptoCurrency Subscribe:** Dealers or subscribers will be able to choose and subscribe to desired cryptocurrency updates such as Bitcoin, Ethereum, Litecoin, Dogecoin, and BCC. The subscriptions will ensure that the users receive the info in due time, according to preference.

2. Publish Cryptocurrency Updates: Publishers are designed to serve as information providers and constantly push updates on cryptocurrency prices or market trends. For simulation, each cryptocurrency is assigned to a particular publisher as if there is only one source of external data in the real world for an environment of disseminating real-time news.

3. Real-time Notifications: On every publisher publishing updated information, notifications are sent out asynchronously to all the subscribing clients, thereby allowing dealers to get updated with hardly any latency. This will enable users to make more informed decisions on trade using the latest available market data.

4. Centralized Login System: The login credentials of the publishers and subscribers are maintained centrally on a server. It enforces secure access to the dealing room system. In this way, the facility of login-in is centralized to improve the security of the users and provides one entry point for managing the different users.

5. GUI for User-Friendliness: For publishers and subscribers alike, a user interface is developed in order to ease interactions within the dealing room. A GUI makes users adept enough to perform elementary tasks such as logging in, subscription to cryptocurrency updates, and latest news, among other added functions, thereby making the system more friendly and approachable.

System Overview

This system is designed based on three main services: the Publisher, Subscriber, and Dealing Room. The Publisher service represents various providers for each cryptocurrency data, where each publisher continuously posts market updates to a central bulletin board. Each publisher would represent a single cryptocurrency and update the system with the latest values or relevant information related to that cryptocurrency. The updates would then be kept in a central server that works as an intermediary between publishers and subscribers.

The Subscriber service represents the dealers who can subscribe to updates on any particular cryptocurrency. A subscriber may also subscribe for new topics and unsubscribe from getting notifications on any particular cryptocurrency. The center server works as a dealing room, ensuring that updates only reach relevant subscribers through asynchronous notifications. This decoupling in communication style lets updates be

efficiently distributed in real time while decoupling the publishers from subscribers directly.

The Dealing Room system [supported by a persistent bulletin board] provides indirect communication by collecting data from various publishers and then distributing it to interested subscribers. Applying an event-driven approach, the system guarantees updates become available to all subscribed clients whenever a new cryptocurrency update is published. The system is designed to mimic modern-day financial dealing rooms' infrastructures that have real-time market data flowing from multiple sources into the diverse end-users in a transparent manner.

Anticipated Results

The expected outcome of this project is an enhanced learning regarding the development of distributed systems using indirect communication and practical experience in the usage of Publish-Subscribe model using Java. This will provide a chance for evaluation of the benefits and drawbacks involved with the use of event-driven, asynchronous notification system in a distributed environment. Beyond that, work on an up-to-date user interface for a dealing room itself exhibits the importance of usability and interactivity in complex systems in the case of real-time data flows.

In a nutshell, this project presents the realization of a scalable and secure information-dissemination system about cryptocurrencies. The work applies the distributed Application Programming Interfaces of Java in order to show exactly how indirect communication models, such as Publish-Subscribe, are able to efficiently support the sharing of data in financial trading applications. In this hands-on project, we seek to develop the necessary skills and knowledge to design robust distributed applications that meet the high demands of real-world trading environments.

DATA STRUCTURE AND ALGORITHMS

The publish-subscribe cryptocurrency dealing room system is based on various key data structures chosen for their efficiency in storing, retrieving, and managing the data with respect to subscriptions, information on Cryptocurrencies, and user sessions. The subsequent sections detail the major data structures used in this program, along with their respective roles.

1. *HashMap*

The HashMap data structure is used in several classes to maintain mappings between certain identifiers and their information. HashMap enables efficient retrieval and storage, whereby most get and put operations can be performed in constant time.

- **cryptoMap**: It is a map stored in **DealingRoomServer**, which holds cryptocurrency articles for every variety of cryptocurrency. Each entry in **cryptoMap** is a map from a cryptocurrency name String to an **ArrayList** of **CryptoObject**, holding all the articles about a cryptocurrency. This allows for quick access to all articles for a given cryptocurrency.

```
HashMap<String, ArrayList<CryptoObject>> cryptoMap;
```

- **SubscriptionList** in **DealingRoomServer**: it maps every topic of cryptocurrency to a set of usernames that subscribe to the update of the concerned topic. This allows for efficient subscription: given a cryptocurrency, retrieve all the users that shall be notified.

```
HashMap<String, Set<String>> SubscriptionList;
```

- **DealingRoomServer clientOutputStreams** This maps each client's **PrintWriter** to their usernames so the server will be able to broadcast any message/update to specific clients by referring to their output stream.

```
HashMap<PrintWriter, String> clientOutputStreams;
```

- **publisherDatabase, subscriberDatabase**: In the implementation class **LoginServiceImpl**, these are **HashMaps** that store publishers' and subscribers'

usernames and passwords. This structure would support an authentication process whereby the system may efficiently verify users.

```
HashMap<String, String> publisherDatabase;
```

```
HashMap<String, String> subscriberDatabase;
```

2. *ConcurrentHashMap*

- The **ConcurrentHashMap** is utilized in dealing with IP addresses within **DealingRoomServer**, ensuring thread-safety, which is crucial when one has a lot of clients with operations that are multithreaded.
`ConcurrentHashMap<String, Integer> ipMap;`
- **ipMap** in **DealingRoomServer**: Every username is mapped onto an IP address - Integer - of the user session. Thus, by using the IP address, the server uniquely identifies clients and shares efficient, thread-safe access.

3. *ArrayList*

The **ArrayList** data structure is supposed to be used principally for maintaining lists of **CryptoObject** instances or dynamic subscribers gathered under each topic.

- **Subscriptions/Subscriber** In **SubscriptionList** Sets of topics are the values in the **SubscriptionList**, each has a `Set<String>` which holds usernames of subscribers who subscribe to that crypto currency. It would allow subscription lists to be dynamic.
`Set<String> subscribers = new HashSet<>();`
- Articles in **cryptoMap**: In **cryptoMap**, **ArrayList** represents articles for every kind of cryptocurrency. **ArrayList** is a dynamic array; hence, it resizes when the number of articles increases or decreases upon adding/removing articles.
`ArrayList<CryptoObject> articles = new ArrayList<>();`

4. Set

In the **SubscriptionList**, the Set data structure is utilized to store subscriber usernames to prevent duplication and to ensure that lookup procedures while maintaining subscription lists are extremely effective.

- **SubscriptionList** Subscriber Lists: It keeps, for every subject in cryptography, a Set<String> of the usernames. This ensures that each user is listed once in every subject, and there isn't the possibility of duplicate subscription.

5. Class-Specific Data Structures

In addition to the collections enumerated above, many of the classes use bespoke structures and/or field attributes to manage a variety of types of data unique to their function within the system:

- **CryptoObject**: Each **CryptoObject** shall be used to store an article's information in terms of cryptocurrency. It shall contain fields such as headline, content, **topicName**, price and **cryptoName**. These fields store core information of the articles which may later get fetched to show or notify. `CryptoObject article = new CryptoObject(headline, content, topicName, price, cryptoName);`

Conclusion

These data structures will, together, set a solid and performance-efficient foundation on which to handle subscriptions, updates of cryptocurrencies, user sessions, and articles with grace. Employing HashMap will enable fast lookups and mappings; **ArrayList** and Set will efficiently support dynamic and unique subscribers and articles collections, respectively. **ConcurrentHashMap** adds thread safety for IP tracking, another key aspect of the publish-subscribe architecture where many clients are supported at once.

PROBLEM SOLVING – DEALING ROOM SERVER

DealigRoomServer represents a core component serving any kind of information regarding cryptographic currency and interactions with clients. It is designed to handle subscriptions, publish real-time crypto news updates to subscribed clients, and manage user IP addresses and subscriptions. Further, some basic approaches to the solution, which have been implemented while developing this server to execute all the assigned tasks effectively, are given below.

1. Handling Real-time Crypto Updates

Problem: The server will broadcast cryptocurrency updates to all the connected clients that subscribe to given topics.

Solution: We utilized a cryptoMap HashMap<String, ArrayList<CryptoObject>> data structure that maps each crypto topic to a list of relevant CryptoObject articles. It allows the server to efficiently access relevant articles and to distribute these to subscribing clients in the event of a new update. As an example of this, the receiveCryptoObject() method in the server processes received updates by looking up the given topic and then spreads the relevant CryptoObject to all subscribers of that topic.

2. Subscription/Unsubscription Requests by Users

Problem: The clients should have the possibility of dynamic subscription and unsubscription to various cryptocurrency topics.

Solution: A SubscriptionList is a HashMap<String, Set<String>> that maintains for each topic its set of subscribing usernames. The methods subscribe add or remove users from a particular topic's subscription list. In this way, methods can remember who is subscribed to what, so that updates will only be dispatched to users who have expressed such interest in those updates.

3. Effective Management of Client Communications

Problem: There should be a scalable system of communication amongst the clients such that, without any redundancy, messages reach all the concerned clients.

Solution: A clientOutputStreams HashMap<PrintWriter, String> is utilized for storing the output streams of all connected clients with their respective usernames. Through utilizing this map, the server can directly send messages to the clients instantly by retrieving the PrintWriter object of each client from it. Such a structure is utilized within the sendHelloToAllClients() method for broadcast upon startup or sending new crypto updates with huge efficiency.

4. Registering and Managing the User's IP

Problem: Because any number of users might connect over different networks, the server has to keep track of the clients' IP addresses for registration and also for security.

Solution: The ipMap is a ConcurrentHashMap<String, Integer>, which is used for maintaining user IP addresses along with their corresponding usernames. IpRegister() updates this map every time one gets logged in, hence ensuring real-time management of IP data in a thread-safe way. This concurrent map copes well with several updates by clients with no sacrifice for concurrency or thread safety.

5. Resilience and Recovery Using Data Persistence

Problem: Users' subscriptions and Crypto articles should be preserved even when any server restarts or crashes.

Solution: Methods like saveSubscriptionListToFile() and saveCryptoMapToFile() allow periodic write-out of SubscriptionList and cryptoMap respectively to external files. In the case of server restart, the method loadCryptoMapFromFile() reloads such data to recover the past server state and thus minimize losses.

6. Support for Scalability and Concurrency

Problem: To handle multiple clients simultaneously, the server must support concurrent access to its data structures.

Solution: Here, the data structure used is ConcurrentHashMap. Such a data structure provides thread-safetiness by design and doesn't require explicit synchronization. Thus, this reduces bottlenecks when there is a need to serve many clients' requests in parallel. In brief, the functionality of the DealingRoomServer component, as described above, uses a set of comfortably structured data management techniques and methods that handle real-time update, user subscription, and persistence storage. Thus, it is made potent and

scalable enough for maintaining cryptocurrency-related data within a client-server environment.

PROBLEM SOLVING – CLIENT

CryptoPublisherClient is the client-side portion, which will allow the user to have the possibility to interact with the server: subscribe to cryptocurrency topics, get updated, and publish crypto articles. The functional requirements of the client, which the implementation is going to support, include smooth and interactive user experiences, ranging from authentication of the user, subscription management, article publishing, and many more. The succeeding sections present the major problem-solving strategies adopted in implementing effective features.

1. Authentication and Role Management

Problem: providing different types of users, such as publishers and subscribers, with secure authentication to access their accorded functionalities.

Solution: The CryptoPublisherClient class has a method called authenticateUser() that interacts with the LoginService at the server level for verification of credentials. This way, a user can enter the system provided that the username and password match those that are recorded on the server. It also differentiates between publisher and subscriber by returning the getUserType and configures the UI according to the role and the role, allowing or not access to the publishing or subscription interface, respectively.

2. Crypto Article Publishing

Problem: The publisher should provide a simple interface to publish new cryptocurrency-related articles and share them with subscribed users.

Solution: It provides a createArticleUI() method for publishers to input the headline, content, topicName, price, and cryptoName for a new article. Then, the created article would be passed along with its CryptoObject from the client's receiveCryptoObject() method to the server. In this way, it ensures that the articles are created in the proper format and delivered to the server to distribute to the subscribers.

3. Subscribe to Real-Time Notifications

Problem: The subscriber should be capable of controlling his subscription dynamically, i.e., subscription and un-subscription for specific topics.

Solution: The functions `subscribe()` and `unsubscribe()` provide a simple option for the user to add or remove topics from his subscription lists. The function is going to call a client-side `subscribe/unsubscribe` method on the server that updates its subscription lists. The described strategy enables the server to send updates related only to each particular user, avoiding useless network traffic and, therefore, improving the user experience.

4. Fetch and Display Crypto Articles

Problem: Subscribers want to view a list of crypto articles, based on the topics they selected.

Solution: The `getArticles()` method fetches the articles according to the topic selected from the `getArticleList()` of the server. This will ensure each client may get the articles of his or her interest. There will also be a basic UI in order to visualize those articles and make the interaction easier for the users by showing just the most relevant and timely articles.

5. IP registration and network handling

Problem: The server must maintain every client's IP address for proper communication and administration.

Solution: The `IpRegister()` method is called during login to register the IP address of each user with the server. This is then set on the server's `ipMap`, allowing tracking of a user's location and opening up a safe channel of communication. This feature has so far been very helpful in remote monitoring, where the server needs to smoothen the configuration of networks.

6. A friendly user interface

Problem: The application should be exposed via a GUI to facilitate publishing and subscription activities with ease.

Solution: `publishGUI` returns the interface through which the publisher will fill out details regarding an article. Also, `showDashboard` shows different features that a publisher or subscriber can access depending on `userType`. Such interfaces will be able to interface the user with the system without much idea of technicalities.

7. Error Handling Support and Reconnection

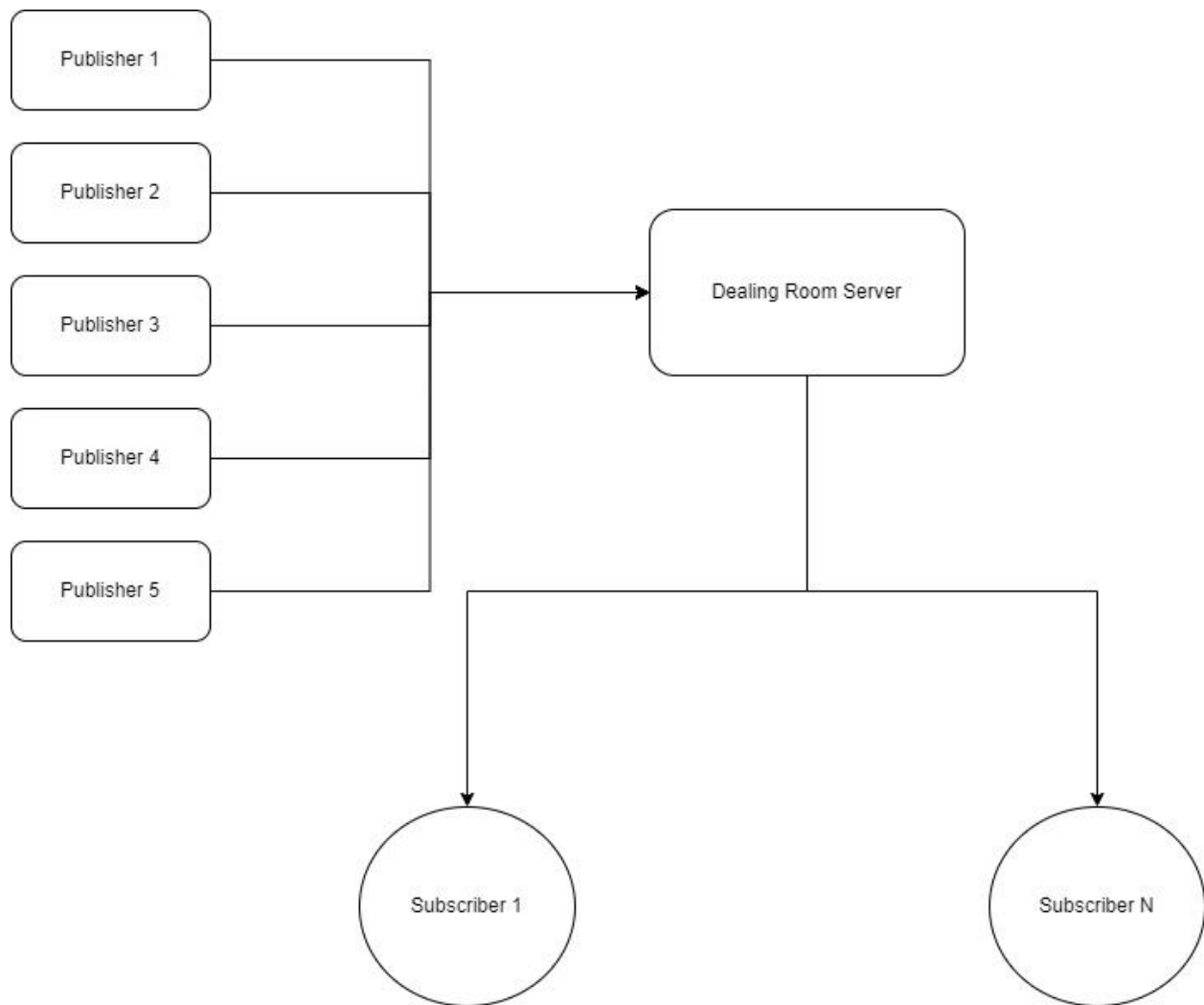
Problem: Network disruptions or server downs can disconnect clients from the server.

Solution: The client handles errors eloquently, especially if there is some interaction with the server. One may introduce retry mechanisms for lost connections or server-side problems or inform the user by appropriate messages. In this way, the client will be more robust for the user and will not include the implications of temporary problems in the network.

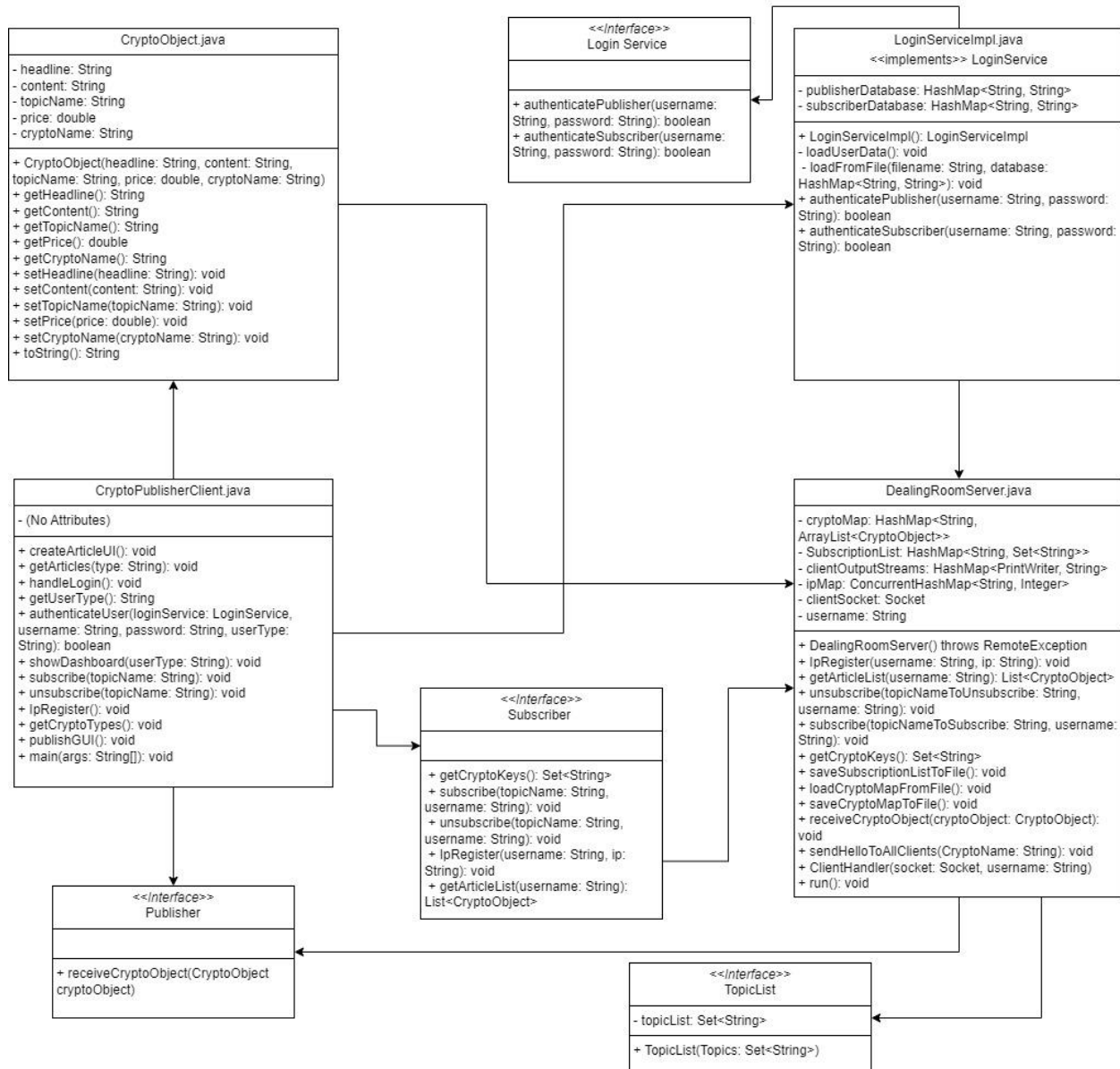
8. Conclusion

In short, CryptoPublisherClient, during its design, takes into consideration several problem-solving techniques that make it work in easy interaction with the server: quick user authentication, fast publication of articles, subscription management, and a user-friendly GUI. The approach hence will make it robust, secure, and more user-oriented for better user engagement with live cryptocurrency data.

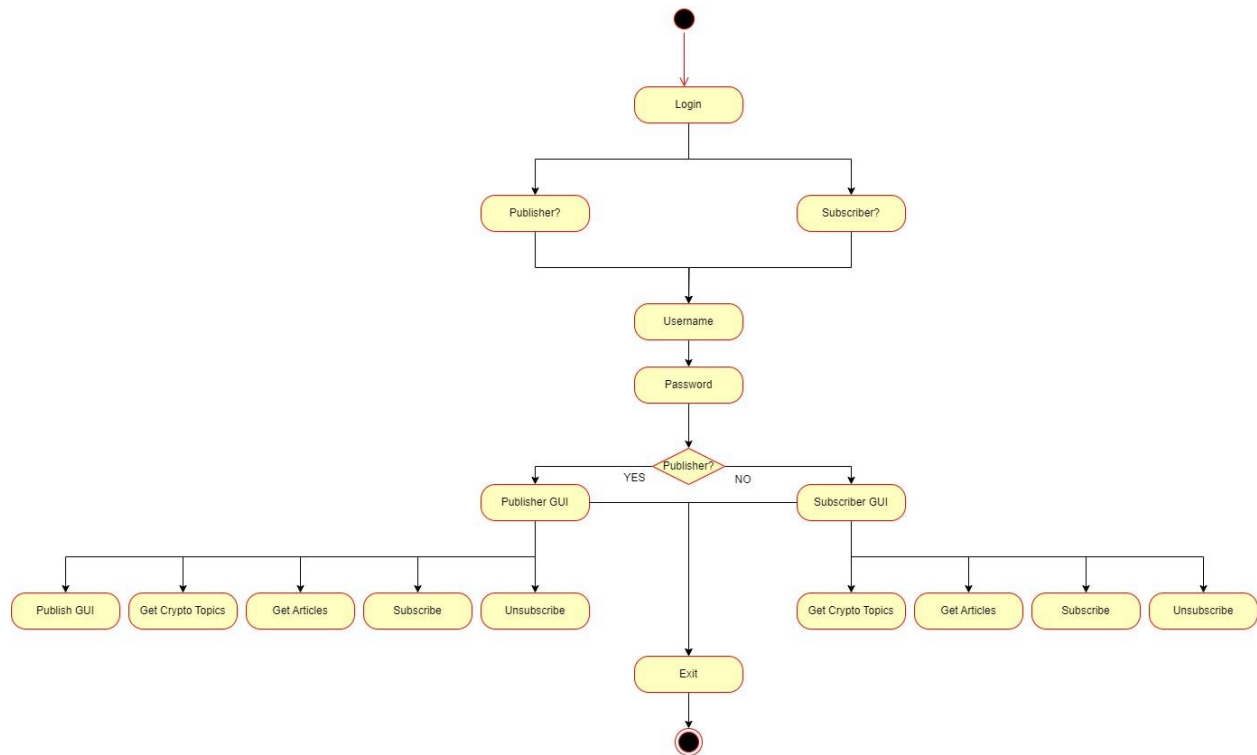
DIAGRAMATIC REPRESENTATION OF THE SYSTEM



Basic Conceptual Diagram



UML Class Diagram

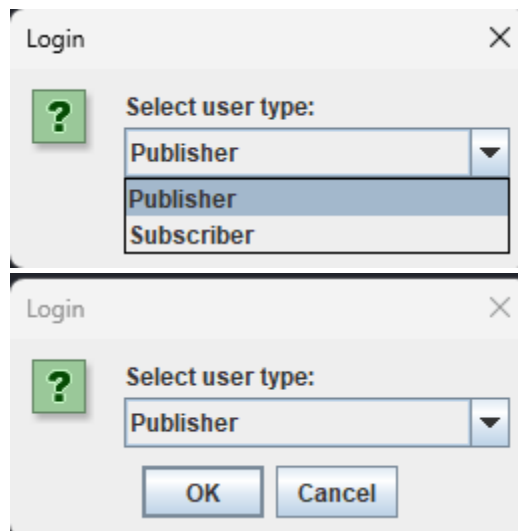


UML Activity Diagram

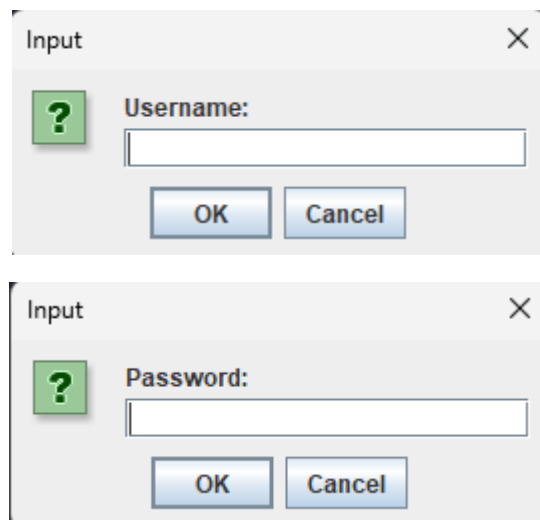
SCREENSHOTS

```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\umaro\OneDrive\Desktop\MSCS\SEM4\A05\A05 Project 2\Code> java DealingRoomServer
Server IP: 10.0.0.239
HashMap loaded from file.
Subscription List loaded from file.
Umar/10.0.0.239
Server is listening on port 10655...
|
```

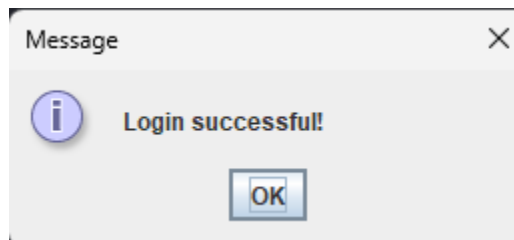
Dealing Room Server Running



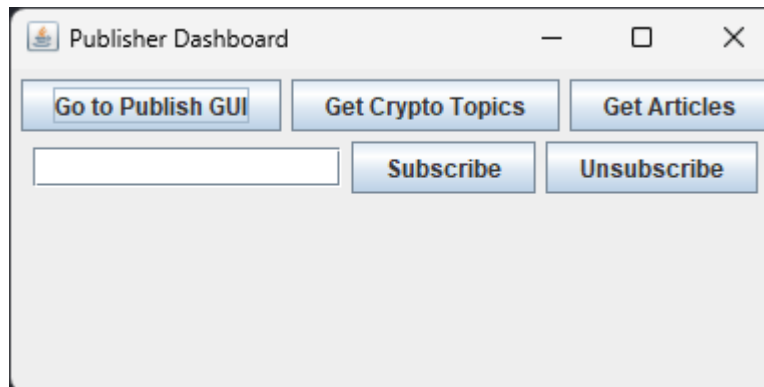
Subscriber/Publisher dropdown menu



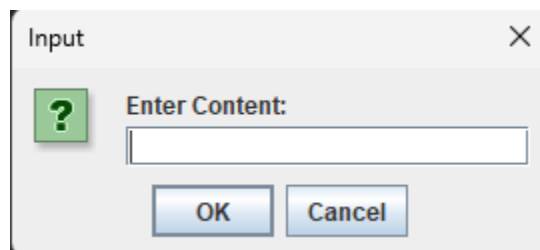
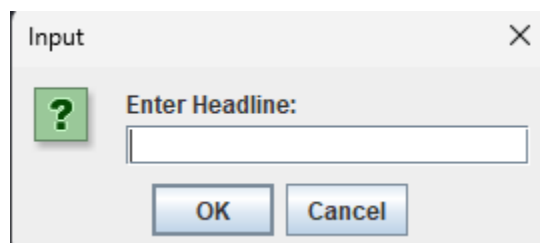
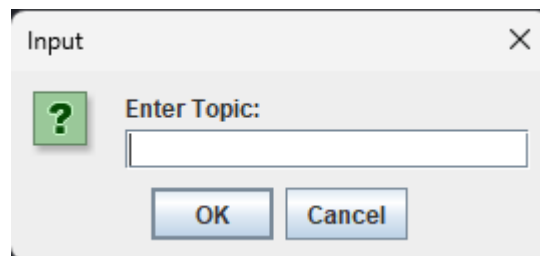
Username / Password Input Fields

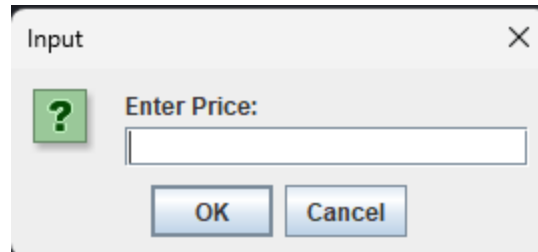


Successfull Login Prompt



Publisher Dashboard

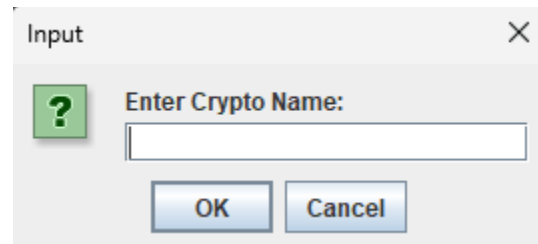




Input

Enter Price:

OK Cancel

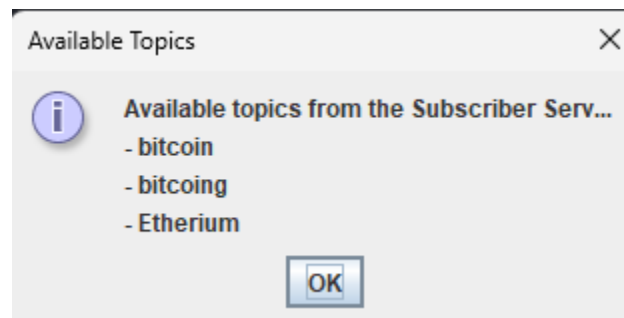


Input

Enter Crypto Name:

OK Cancel

Fields to Publish and Article



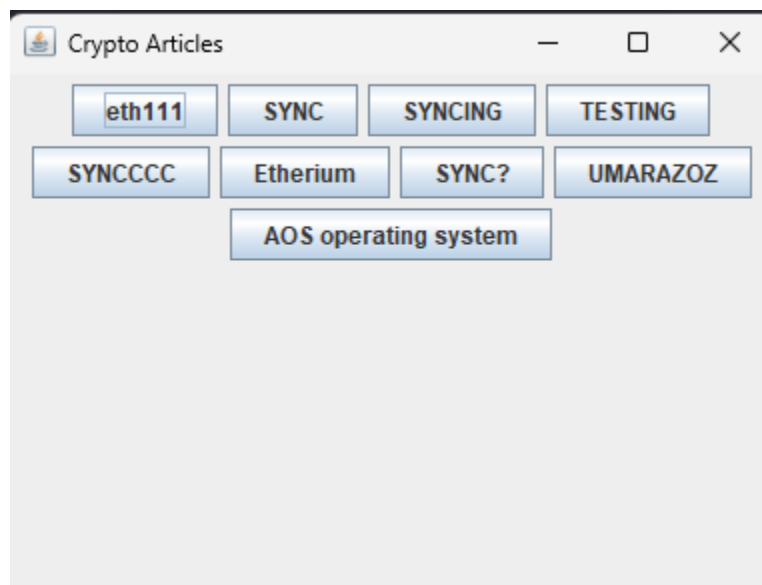
Available Topics

Available topics from the Subscriber Serv...

- bitcoin
- bitcoing
- Ethereum

OK

Available Topics



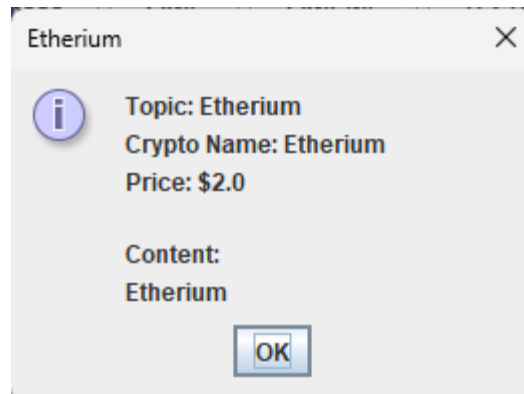
Crypto Articles

eth111 SYNC SYNCING TESTING

SYNCCCC Ethereum SYNC? UMARAZOZ

AOS operating system

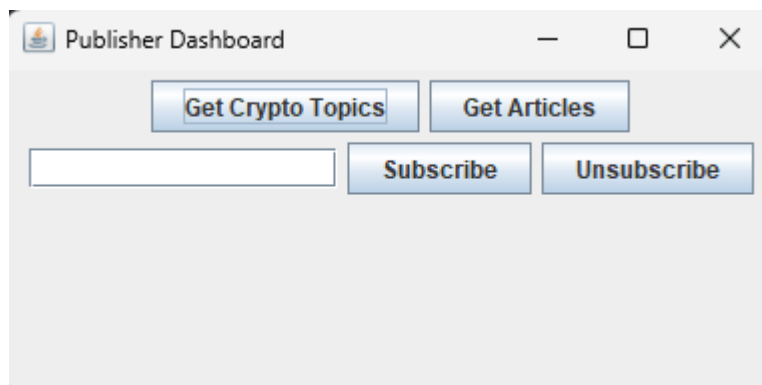
Subscribed Article Fetching



Ethereum Crypto Object

```
PS C:\Users\umaro\OneDrive\Desktop\MSCS\SEM4\AOS\AOS Project 2\Code> java DealingRoomServer
Server IP: 10.0.0.239
HashMap loaded from file.
Subscription List loaded from file.
Umar/10.0.0.239
Server is listening on port 10655...
Client connected: 10.0.0.239:54889
Waiting for username...
IP Registered: {10.0.0.239=0, 10.0.0.239:54889=54889}
Received from client: Client connected
Waiting for username...
Received from client: bitcoin
Sending Crypto Keys to client...
Subscriber added to topic: bitcoing
Subscription List saved to file.
Subscriber removed from topic: bitcoing
Subscription List saved to file.
█
```

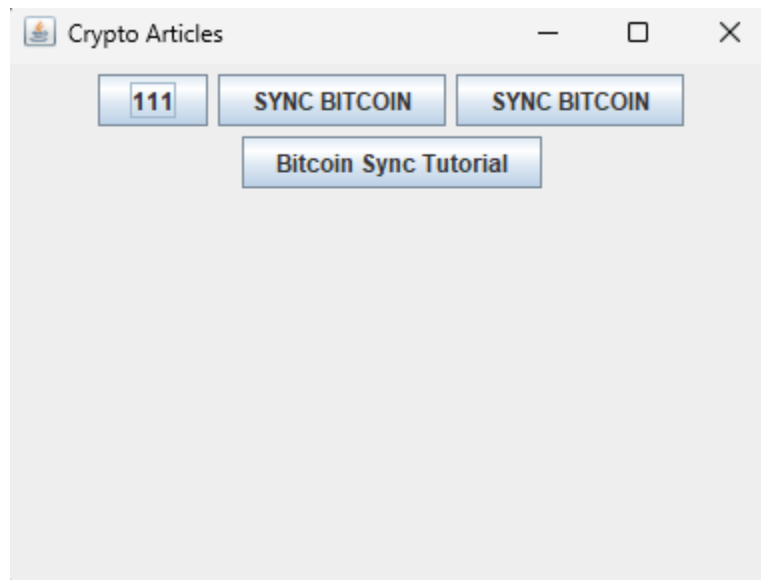
Dealing room after subscribing and unsubscribing



Subscriber Dashboard



Publisher Article Release Notification



After Clicking ok on notification, new article window is opened with recently released article.



CONCLUSION

The design and implementation of DealingRoomServer and CryptoPublisherClient are done in such a way that the system is scalable, making management and publishing of cryptocurrency-related content in real time easily accessible. Due to the reusability provided by RMI, the server and client can talk easily: the publisher publishes new crypto articles, and the subscriber gets timely updates according to their interest.

DealingRoomServer plays an important role in the solution of some of the major problems such as subscription system dynamism, real-time updates of articles, and doing this with efficiency at the client-server level. Tracking subscribers, keeping crypto-articles updated, and updating connected clients. its ability to do all these plays a critical role in ensuring users get the most current and relevant information.

CryptoPublisherClient is an easy-to-use client-side abstraction not only for publishers but also for subscribers: log in, publish article, and manage subscriptions. It splits the publisher and subscriber-related functions apart, giving each of the user types exactly what they need to do their jobs. It also separates IP registration and network management so that the user does not need to worry about the connection.

Taken together, these components solve a few key issues like real time communications, ease of use interfaces, and data consistency among the client and the server. The RMI applied will ensure smooth and secure communication between the client and server; the modular architecture provides room for flexibility and scalability.

In the end, the system comprehensively covers the issue of cryptocurrency content distribution between publishers and subscribers. Further development needs to be performed for more security, enhancing performance for larger user bases, and extending for more crypto-related features or integrations. The proposed architecture is built in such a way that it can have a solid foundation for further expansion and adaptation to new requirements for users and market conditions.