User Profiles and Recommendations on Voice/Audio-based Social Networks Using Neo4j

Institution: School of Computer Science and Informatics, Cardiff University

Author: Zhe Deng

Supervisor: Alia I Abdelmoty

Degree: MSc Computing

Module Code: CMT403

Date: 06/09/2024

Acknowledgements

I would like to express my gratitude to my supervisor, Alia I Abdelmoty. She helped me a lot when I faced challenges in this project. Her continuous support and patience have encouraged me to observe problems from my project and find solutions for them.

I also want to appreciate my friends and parents. Their care and encouragement enabled me to overcome difficulties during my course and the project process and complete this project.

Abstract

With the rise of audio social networks, how to build user profiles and recommendation engines for them has attracted considerable interest. Graph databases have significant advantages in social networks and recommendation systems, and Neo4j is one of the most popular graph databases. This project uses Neo4j to build an audio social network model and simulate the process of making recommendations in this system. This system allows users to participate in audio rooms and simply interact with other users, and uses specific algorithms to make recommendations to users.

Contents

List of Figures

List of Tables

1. Introduction

Audio-based social networks have become popular since Clubhouse launched in 2020. Audio-based social media is a kind of social networking platform in which members mainly communicate by audio instead of text and images. It has become a new trend in social networking: text is not enough to satisfy people's desire for human connection, and video is too formal for people to communicate with friends(jeremiah_owyang 2021). Building user profiles on audio-based social networks and recommending the "rooms" that users may be interested in has been important since audio-based social networks became popular.

With the explosive growth of information on the Internet, recommender systems are valuable tools that help users overcome information overload (Doh et al. 2022). Recommender systems can be classified into three broad groups: content-based systems, collaborative filtering systems, and hybrid systems (Guo et al. 2022). Content-based recommender systems mainly use features of items to find their similarity and then to recommend similar items which have the same features to users, while collaborative filtering-based recommender systems focus on similarity measures between users and/or items (Leskovec et al. 2019). Hybrid recommender systems combine multiple technologies to compensate for the shortcomings of a single recommendation algorithm. With the development of technology, new recommendation methods have been proposed, and knowledge graphs become an important resource to support recommender systems (Chicaiza and Valdiviezo-Diaz 2021).

A knowledge graph is a multi-relational graph composed of entities and relations, which are regarded as nodes and different types of edges (Ji et al. 2022). In a basic recommender system, items and their features, and users and their side information can be mapped into the knowledge graph to represent the relations between users and items and capture user preferences. Neo4j is a popular graph database for building knowledge

graphs. Unlike relational databases, neo4j supports named, directed relationships between nodes, which give a rich semantic context for the data. This makes it suitable for formulating real-time recommendations. On the other hand, the recommendation results of the knowledge graph-based recommender system are explainable (Guo et al. 2022). In this project, I will use Neo4j to build a Twitter Space data model and a recommendation engine based on this model and evaluate whether it is an effective way to build a recommendation engine by using neo4j.

1.1 Aims

This project aims to build user profiles and recommendations for Twitter Space using a graph database to store audio-based social network data. This was achieved by investigating three main questions:

- How to store social network data in a graph database and build user profiles for an audio-based social network platform.
- How to build a recommendation engine based on the data model.
- How to improve the performance of the recommendation engine and this system.

To answer the questions above, this project can be divided into two parts:

Part 1: Twitter Space Data Model Design

This section mainly concentrates on designing a graph database model to store user profiles and space data of Twitter Space and developing an application to mimic its basic functions. To develop this model, it is important to investigate which data Twitter Space has stored and how to store these necessary data in Neo4j to ensure the implementation and efficiency of basic functions.

Part 2: Build a Recommendation Engine for the Twitter Space Data Model

In this section, a recommendation method needs to be proposed based on the

model designed in part 1. Then, this recommendation method needs to be implemented in the application and tested for efficiency. Finally, consider how to improve the recommendation engine based on the results.

1.2 Objectives

I need to accomplish the following objectives in this project:

- Study how to use Neo4j and its programming language Cypher.
- Understand how Neo4j can be used as the database in an application.
- Consider the functions that the application needs to achieve, design an appropriate Twitter Space data model for it, and evaluate the efficiency of this model.
- Build a recommendation engine by choosing appropriate methods and evaluating its efficiency.

2. Backgrounds

2.1 Graph Database

Recently, graph databases have gradually occupied an important position in software development; most internet giants, such as Facebook, use graph databases to store enormous amounts of data (Paul et al. 2019). A graph database stores data through the arrangement of nodes, edges, and relationships. Nodes can represent entities, which can be real-world objects or abstract concepts. Edges can represent relationships between nodes; they connect nodes in the graph. Through edges, every node can be directly linked to its neighbouring nodes and acts as an index of its neighbouring nodes, which is called index-free adjacency (Pokorný 2015). Figure 1 shows an example of graph database models in audio-based social networks. In this graph, users, spaces, and topics are represented as nodes, each containing additional data referred to as properties. The relationships between nodes are represented by directed edges, and the edges also

have properties.

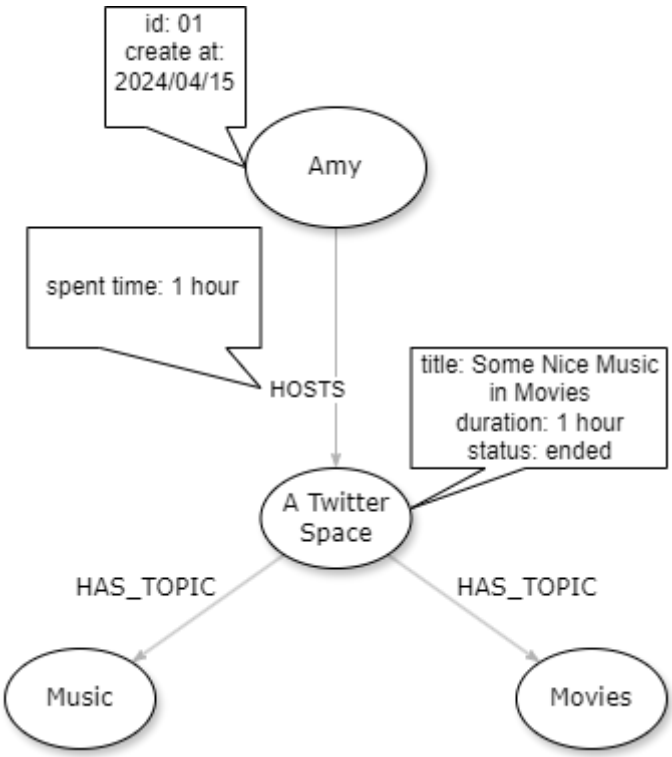

Figure 1: A simple graph database example

Graph databases have significant benefits compared with traditional relational databases (RDBMS). They can store and query data with complex relation structures directly without joining multiple tables, which improves query performance. Graph databases have huge potential when dealing with highly connected data models like social networks and recommender systems. Neo4j is the most popular graph database in the world. Many companies, such as Walmart, use Neo4j to serve up real-time recommendations(Webber 2021). In this project, Neo4j will be used to build user profiles and a recommendation engine for audio-based social networks.

2.2 Audio-based Social Networks

As the introduction mentions, audio-based social networks have gradually thrived in recent years. The existing popular audio-based social network platforms mainly include Clubhouse, Twitter Spaces, and Facebook Live Audio Rooms, each with its

own features. I will mainly analyse the components of Twitter Space and which aspect to build user profiles from.

2.2.1 The Main Components of Twitter Space

The main components I identified for Twitter Space include:

- The basic social interaction part allows users to communicate in the audio chat room. Users can listen to speakers in the Space and express their views on the speakers' remarks by sending specific emoticons; they can also request to speak in the Space and write a reply to the Space or share this Space with others.
- The search function allows users to search Spaces or users most related to the input keywords or phrases.
- The recommendation section recommends the trending live Spaces, the Spaces users may like based on topics they are interested in and historical preferences, and the Spaces that the users' followed contacts will participate in.
- The notification part allows users to receive a notification when their space is about to start. Users can create a Space, choose topics about the space and its scheduled time, and then invite other users to participate.

The basic social interaction features are the most important parts of Twitter Space; they are also the core features of audio-based social networks. They allow multiple users to communicate through audio and express their opinions on speakers' views without delay. This function requires high concurrency and real-time audio data transmission. This project concentrates on building a data model to simulate the basic function of Twitter Space and write a recommendation engine, therefore this project does not involve real-time transmission of audio data. Furthermore, Users can follow or block other users, or view the profiles of users in the same Space.

The search function requires both query efficiency and result relevance when users search for content. In the Twitter Space application, after users enter a keyword, the

initial results displayed are live Spaces whose titles or hosts contain the keyword. This is followed by upcoming Spaces whose titles or hosts contain the keyword, and users can add these upcoming Spaces to their calendars.

The recommendation is also an essential part of audio-based social networks; it can help users quickly find the content they may be interested in. The recommended Spaces in Twitter Space can be divided into five groups: the trending Spaces, Spaces related to the topics in the user's preferred topics list, Spaces that users' followed users will be turning in, Spaces recommended based on user historical data and Spaces hosted by the most popular hosts. These recommendations are mainly based on the strength of user profiles; hence, building user profiles is crucial for personalised recommendations.

2.2.2 User Profiles on Twitter Space

Since Twitter Space is a branch application of Twitter, the user profiles will integrate the user data both in Twitter and Twitter Space. Below are user profiles that Twitter and Twitter Space may have:

Basic information	
User ID	User's unique identifier on the platform
Username	A unique username on the platform
Name	It is not unique but also provides identifying information
Bio	A brief introduction of the user on the platform
Location	User's current location
Website	User's personal website
Birth date	User's birth date
Phone	A mobile number that can be linked to the account
Email	An email address used during account registration
Country	The registered location.
Interests	Selected topics which the user likes during registration
Join Date	The date that the user registered this account

Social relationships	
Following	Accounts that the user followed
Followers	Accounts that follow the user
Blocked Accounts	Accounts that the user blocked
User activities	
Posts	User's personal posts
Likes	Posts that the user liked
Replies	User's replies for Posts
Joined Spaces	Spaces that the user has listened to and the time the user spent separately in these Spaces
Hosted Spaces	Spaces that the user has hosted and the time the user spent separately in these Spaces
Moderated Spaces	Spaces that the user joined as a speaker and the time the user spent separately in these Spaces

Table 1: The probable user profiles for Twitter and Twitter Space

User should choose their interested topics when registering an account on Twitter, which is a strategy to recommend content for new users without preference data. The user preference data can be enriched with the growth of user activities; these data need to be found by the recommendation engine.

2.3 Recommendation Systems

A recommender system, or a recommendation engine, is a software tool which can provide suggestions of items that specific users are most likely to be interested in (Ricci et al. 2022). The recommendations are typically personalized; different users can benefit from customised suggestions. To predict the most suitable items for users, the recommendation engine needs to collect information about users' preferences, which may be explicit product ratings or implicit indicators that need to be inferred by interpreting the actions performed by the user when interacting with the system (Ricci

et al. 2022). After the recommendation engine recommends items to users, they can choose whether to accept these recommended items, and then they may provide explicit or implicit feedback that will be stored in the system database to provide more accurate recommendation services in the future.

According to the techniques used, traditional recommender systems can be divided into three groups: content-based recommender systems, collaborative filtering recommender systems, and hybrid recommender systems. I will introduce these three traditional recommender systems in the following.

2.3.1 Content-based Recommender Systems

Content-based recommender systems concentrate on items' properties. The similarity of items is calculated based on the similarity of their properties. Therefore, in content-based recommender systems, it is essential to extract the item's features and build a feature set for the item.

Some values of features are apparent. For example, consider a movie recommendation system with ratings from users; the features of a movie can include the directors, the set of actors and the genre of the film. Users may prefer movies with their favourite actors and directors or movies with a particular genre, such as comedies(Leskovec et al. 2019).

There are also some features' values which are not obvious. Some document collections, videos, audio files, and images generally lack readily available information to provide sufficient features(Leskovec et al. 2019).

When extracting the features of a Twitter Space, we can use the topics selected by the creator as features. However, a Space also has audio content, and such an audio file does not have readily available information to provide features. A possible effective method is to use tools to transcribe the audio file and then process the transcribed text. To process the transcribed text, we can eliminate the most common words and then calculate the TF.IDF score for the remaining words. After this, we can choose an

appropriate range to get the feature words. For example, we can select words with TF.IDF scores above a specified threshold to include as part of the feature set for the Space. Finally, a set of words is obtained to represent the features of the Space. Then, the similarity between different Spaces is calculated based on a specific similarity calculation method, such as the Jaccard Similarity Coefficient. This is a possible solution to build a content-based filtering recommendation engine for Twitter Space.

2.3.2 Collaborative Filtering Recommender Systems

The collaborative filtering recommender systems mainly recommend items by identifying users with similar interests and recommending items that similar users like. The similarity between two users is calculated by calculating the similarity of explicit ratings on items of the two users, or implicit indicators. Therefore, calculating the similarity is a crucial point in this circumstance. Before discussing how to calculate the similarity, I need to introduce the concept of utility matrix.

In a recommender system, there are two types of important entities: users and items (Leskovec et al. 2019). Users have preferences for these items. Suppose there is a series of user-item pairs, and each pair has a value representing the user's preference for this item; we can build a user-item matrix named a utility matrix based on these user-item pairs. The values in the utility matrix are the users' preference values for these items. Some recommender systems have explicit user ratings for items, so the values in the utility matrix can be the user ratings or processed ratings; some recommender systems do not have explicit ratings, so it is necessary to infer the user's preference for items based on the historical interaction between the user and the system. For example, we can simplify the user interaction between the user and a Twitter clone system to understand this process: if the user views a post and hits a like button on this post, the value in the user-post matrix may be equal to 1; if a user chooses "Not interested in this post", this value may be -1; and in this case, 0 usually means that the user has not viewed this post. The actual way Twitter recommend posts may be more complicated

considering complex user interaction behaviours.

Based on the utility matrix, we need a reasonable distance measurement method to calculate the similarity between two users. For instance, suppose there are two users represented by A and B; their ratings for items are shown in Table 2, with the ratings between 1 and 5.

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9
A				4	5			1	
B				2	1			5	

Table 2: An example utility matrix

As we can see from the table, the two users have opposite ratings on the shared items, so it is important to choose a suitable distance measurement method to make them rather far apart.

The relevant similarity calculation method will be explained in the next section of this chapter.

2.3.3 Hybrid Recommender Systems

Hybrid recommender systems mainly combine different recommendation methods, such as combining content-based filtering and collaborative filtering or combining collaborative filtering and demographic filtering to leverage the strengths of each method. This technique can solve the cold-start problem in a recommender system. When a user has no historical interaction data in the system or has very little interaction data, a collaborative filtering recommender system can’t make accurate recommendations. Therefore, collaborative filtering can be combined with other techniques to overcome this challenge. Hybrid filtering is usually based on bioinspired or probabilistic methods such as genetic algorithms, fuzzy genetics, neural networks, Bayesian networks, clustering, and latent features(Bobadilla et al. 2013).

2.4 The Similarity Calculation Method

The general way to calculate the similarity between samples is to measure the distance between their features. There are many mathematical methods to calculate the distance. In this section, I mainly introduce three coefficients which can represent the similarity between samples.

2.4.1 Jaccard Similarity Coefficient

The Jaccard Similarity Coefficient is an indicator used to measure the similarity between two sets; it is defined as the ratio of the size of the intersection of the two sets to the size of the union; the formula is as below:

$$sim_{jac}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where A and B represent two sets.

The Jaccard Coefficient is suitable for binary sets or discrete data sets. Therefore, when processing data with ratings or other non-binary data, using the Jaccard Coefficient might result in the loss of some important information. On the other hand, it can't provide satisfying results when handling sparse matrices.

There is a generalised Jaccard Similarity Coefficient which may be applicable to continuous data instance, it refer to them as vectors and make use of an inner product, and its formula is expressed as(Levy et al. 2024):

$$sim_{jac}(P, Q) = \frac{P \cdot Q}{||P||^2 + ||Q||^2 - P \cdot Q}$$

Where $P \cdot Q$ represents the dot or scalar product of Cartesian coordinates, $||P||$ and $||Q||$ represents the norm: $||P|| = \sqrt{P \cdot P} = \sqrt{\sum_{i=1}^{n} P_i^2}$, $||Q|| = \sqrt{Q \cdot Q} = \sqrt{\sum_{i=1}^{n} Q_i^2}$.

2.4.2 Cosine Similarity

Like the generalised Jaccard Similarity Coefficient, items or users are modelled as vectors in an n-dimensional space, and the Cosine Similarity is the cosine value of the angle between two vectors. The formula is:

$$sim_{Cos}(A,B) = \frac{A \cdot B}{||A||\,||B||}$$

Where $A \cdot B$ represents the dot product as the same, $||A||$ and $||B||$ represents the norm.

The value of Cosine Similarity is between -1 and 1, the value of Cosine Similarity closer to 1 indicates higher similarity. Cosine Similarity is applicable to discrete data.

2.4.3 Pearson Correlation Coefficient

Pearson Correlation Coefficient is also common in recommender systems(Lathia et al. 2008), it is used to measure the degree of linear correlation between two variables. It is defined as:

$$sim_{Pea}(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

Where X_i and Y_i can represent the different ratings or other values which can indicate user interests in items from two different users, $\overline{X}$ and $\overline{Y}$ represents the mean of the corresponding user ratings.

The value of the Pearson Correlation Coefficient is also between -1 and 1, the closer the Pearson Correlation Coefficient is to 1, the more positively correlated the two items are; -1 indicates a negative correlation, and 0 means no linear correlation between these variables.

The Pearson Correlation Coefficient can be improved based on the data conditions, such as introducing significance weighting to incorporate a degree of reliability into the coefficients(Lathia et al. 2008).

2.5 Existing Solutions for Audio-based Social Networks

In addition to Twitter Space, there are several existing audio-based social network platforms. I will introduce the other most popular platforms in this section.

2.5.1 Clubhouse

Clubhouse is one of the first audio-based social network platforms; its interface is simple and easy to use. Users can invite friends to chat and send voice messages; the message will be directly transcribed into text. Additionally, users can join an audio chat room by searching for keywords and finding the room or joining some social groups named “house” in this application and participating in the group’s audio chat room. However, this application lacks recommendation functions, so it relies more heavily on users actively searching for audio chat rooms that align with their interests.

2.5.2 Twitter Space

Twitter Space is an additional part of Twitter. Users only need a Twitter account to join or host an audio chat room. This means that the number of audio chat rooms may be larger. So, a recommendation engine is required to recommend suitable content to users.

2.5.3 Facebook Live Audio Rooms

Facebook allows public figures and specific groups to host live audio rooms, and there is no limit on the number of listeners. It has some nifty features, such as live captions and notifications when your friends join a room(Carman 2021). However, this application also lacks recommendations for audio-based platforms. It is hard to find a live audio room on the homepage.

As mentioned above, some platforms do not provide live audio room recommendations for users, which may be due to concerns regarding privacy and user experience. However, this may result in users spending more time searching for audio rooms that they are interested in. Twitter Space mainly recommends live audio rooms with trending topics or topics that the user follows. It is necessary to explore an effective method for recommending audio chat rooms.

3. Project Approach and Specification

Since the project's main work is to build a Twitter Space model using a database graph and then a recommendation engine based on the model, in this section, I will mainly introduce the development methodology I used during the development process, the tools I used, the functional and non-functional requirements, the data modelling, the recommendation methods, and the data import methods.

3.1 Development Methodology

Due to the limited development period and fluid requirements, the Agile methodology was applied throughout this project to revise the plan when facing unforeseen challenges or having new findings.

During the development process, I did encounter some situations where I needed to adjust my program and plan. I initially decided to create a combined program with basic Twitter and Twitter Space functions, but I have not found an ideal data model that allows users to quickly access the newest content while also enabling them to view all posts from a specific user, so I temporarily put the Twitter part on hold and focus on the development of Twitter Space. On the other hand, I planned to incorporate users' social relationships for recommendation, but due to time constraints, I was only able to implement a basic recommendation system. The Agile methodology indeed makes me adapt to the continuously changing plan. When I found some shortcomings during the development and test process, or when I had new ideas during development, I made adjustments quickly, and some ideas have not been implemented owing to time constraints; I will explain them in the future work section.

3.2 Development Tools

3.2.1 Python

In order to reduce the complexity of software development process, this project was developed by using Python, and the version is Python 3.12; the web framework used is Flask, and the version is 3.0.3; the Python library used to interact with neo4j on

the back-end is Py2neo, and the version is 2021.2.4.

Flask

Flask is a lightweight web micro-framework written in Python. Its core function only includes some basic web development functions without dependencies on other third-party libraries, so it is flexible and easy to develop a simple web application quickly. Flask also offers strong scalability; it can support the extensions I need to conveniently add the functions. Since I have used Flask to build web applications in previous modules, I'm more familiar with Flask; moreover, this application is used for research and does not need to be deployed to a production environment. I chose Flask for back-end development. I also used some Python libraries that extended Flask's functions during the development process, including Flask-login, Flask-SocketIO, and Flask-APScheduler. Flask-Lgoin is used to manage user login and logout, user sessions and access control; Flask-SocketIO is used to integrate WebSocket functions into Flask applications in order to push messages in real-time to the client; Flask-APScheduler is used to manage timed task scheduling in this project, primarily to notify the creator of the Space when the scheduled time arrives.

Py2neo

Py2neo provides a convenient API to access and operate the Neo4j database. It makes the operations on graph databases more intuitive and also supports all the core functions of Neo4j and the Cypher language. Py2neo is easier for me to use and can support the function requirements of this project, so I finally chose it for development.

3.2.2 Neo4j

Neo4j Desktop

When I chose the Neo4j tools, I tried using both the Neo4j Community Server and the Neo4j Desktop. I finally used the Neo4j Desktop because it is concise and intuitive, and it makes it easier for me to learn Neo4j. On the other hand, this project did not need to be deployed, so I only needed to develop the system in the development environment,

and Neo4j Desktop is suitable for the development process. Therefore, it is enough for me to develop this project by using Neo4j Desktop.

Cypher

Cypher is a programming language designed for Neo4j; its syntax is relatively simple, and its style is close to natural language. Additionally, it can support queries on complex relationships and paths. Cypher is appropriate for building a recommendation engine.

3.2.3 WebSocket

Some functions I designed for this system require data updates in real-time, and the traditional request-response model of the HTTP protocol requires the client to send a request to the server, and then the server can respond, which cannot meet the requirements of real-time data transmission, so I use WebSocket to implement these functions. WebSocket is a computer communications protocol that provides a simultaneous two-way communication channel over a single Transmission Control Protocol (TCP) connection (WebSocket. 2024). It is suitable for developing applications such as real-time chat and online games. Since I use Flask for back-end development, the server uses a Flask extension package named Flask-SocketIO, which simplifies the process of applying WebSocket in a Flask application.

3.2.4 Git

Git is a distributed version control system which is particularly useful in the Agile development process. I use Git to track my progress and save the historical versions. When I fail to develop a new function or fix the bugs, the system can be reverted to historical versions. It can help me manage this project better.

3.3 Requirements

3.3.1 Functional requirements

Register function

The system must allow users to register an account by setting their username and password and confirming the password.

Acceptance Criteria:

- When a user enters a username already in the database, a notification should remind the user that the username exists.
- When a user enters a username that is not in the database, the system should create a user node and store it in the database.
- When the password and confirmed password do not match, the system should remind the user to ensure consistency.

Login Function

The system must allow users to log into their account by entering their username and password.

Acceptance Criteria:

- When a user enters a username that is not in the database, the system should remind the user to register instead of login.
- When a user enters a registered username and the correct password, the user should log into the system successfully and be redirected to the website's home page.
- When a user enters a username that exists in the database with a wrong password, the user should receive a notification that the password is incorrect.

Log out function

The system must allow users to log out after they log into the system.

Acceptance Criteria:

- When a user logs into the system and clicks the logout button, the system

should clear or invalidate the session and authentication data and redirect to the login page.

Create an audio chat room function

The user should be able to create an audio chat room and start the room.

Preconditions:

- The user should have logged into the system.

Acceptance Criteria:

- A Create Space page should be accessible to find when a user wants to create a Space.
- When the user is on the Create Space page, there should be a text area for entering the title, a choice for the user to choose the topics of the Space, and an option for the user to select the time to start.
- When users decide on the topics of the space, there should be a selection model that allows them to choose up to three topics.
- If the user has chosen three topics and wants to add another topic, the system should not add this topic and remind the user that the topics of a Space cannot exceed three.
- When the user clicks the “Create Space” button without choosing a scheduled time, the system should display a reminder message for the user to choose a scheduled time.
- When the user clicks the “Create Space” button with the title and scheduled time, a Space with the information inputted should be created.
- When the scheduled time of the user-created Space arrives, the user should receive a reminder message.

Start an audio chat room function

The user should be able to start the audio chat room created by the user.

Preconditions:

- The user should have created an audio chat room whose status is upcoming.

Acceptance Criteria:

- When the user clicks the “Start Space” button, the space should be started and available to other users.
- A button should be provided to allow the user to start the Space in advance if they wish to do so before the scheduled time.

End an audio chat room function

The user should be able to end the live audio chat room created by the user.

Preconditions:

- The user should be the creator of the audio chat room.
- The audio chat room should have been started.

Acceptance Criteria:

- When the creator clicks the “End Space” button, the Space should be ended, and all users should leave and set a leave time for the relationships between users in the Space and the Space.
- When the creator ends the Space, other users in the Space can’t set a leave time for the relationships between them and the Space.

Join and moderate an audio chat room function.

The system should allow users to join or moderate a Space.

Preconditions:

- The user should not remain in a Space.

Acceptance Criteria:

- The user can choose to join or moderate a Space.
- If the user chooses to join the Space for the first time, the database should create a “JOINS” relationship between the user and the Space, and the webpage of this Space should add the user to the listener's list.
- If the user chooses to moderate the Space for the first time, the database should create a “MODERATES” relationship between the user and the Space, and the webpage of this Space should add the user to the speaker's list.
- After users join or moderate Space, they cannot join, moderate or host another Space unless they leave this Space.

Leave an audio chat room function

The system should allow the user to leave the audio chat room.

Preconditions:

- The user must have joined the room and remained present without leaving after their most recent entry.
- The user is not the creator of the Space.

Acceptance Criteria:

- When the user clicks the “Leave the Space” button, a leave time property should be set for the relationship between the user and the Space.
- After users leave the Space, they should be able to join or moderate another Space or rejoin it.

Follow, unfollow, block and unblock function

The system should allow users to follow, block, unfollow, and unblock other users.

Preconditions:

- The user must be logged into the system.

Acceptance Criteria:

- The user should be able to view other users' profiles and follow or block them.
- The user can only check their own blocklist.
- The user can unfollow/unblock other users they followed/blocked.

Recommend Space function

The system should be able to recommend Space for users.

Preconditions:

- The user should have logged into the system.

Acceptance Criteria:

- Users with historical activity records can get recommendations based on their historical activity.
- Users who did not participate in Spaces can get basic recommendations such as the most popular Spaces.

3.3.2 Non-functional requirements

Security

- Users' sensitive data, such as passwords, should be encrypted before storing in the neo4j database.
- The main application can be accessed only after logging into the system.

Performance

- The recommendation page should not take long to load; the response time should best be within three seconds.

Usability

- It should be easy for users to find and access the system's functions.
- When users fail to perform some operations in this system, there should be clear instructions to remind users of what went wrong.

Reliability

- The recommended results should be accurate and reflect users' preferences to some extent.
- The system should handle data writing, reading, and updating correctly according to users' operation.

Portability

- The system should work properly on major browsers with consistent performance and layout.

Maintainability

- The system architecture should be reasonable, and the code should be clear and easy to understand, making it easy to test and maintain.

3.4 Data Modelling

In order to build a basic data model for Twitter Space, we need to identify its entities and relationships. Since I could not get actual data from Twitter Space, analysing the possible data in the Twitter Space database is essential. In the previous part, I analysed the possible user profiles and current functions of Twitter Space. In the following section, I will propose a possible data model and explain my idea of modelling this way.

As shown in Figure 2, I designed the data model using the star topology. This model directly creates relationships between two related nodes, making it easy to implement in a database. I set some properties for the Space node and the relationship

between the user and the Space to record the user's interaction with Space and determine how much the user is interested in this Space. Since I don't have detailed data about Space, such as the content of the Space audio, I use topics as the feature of Spaces. Each Space would have up to three topics, and the topics of the Space may represent the participating users' interests.

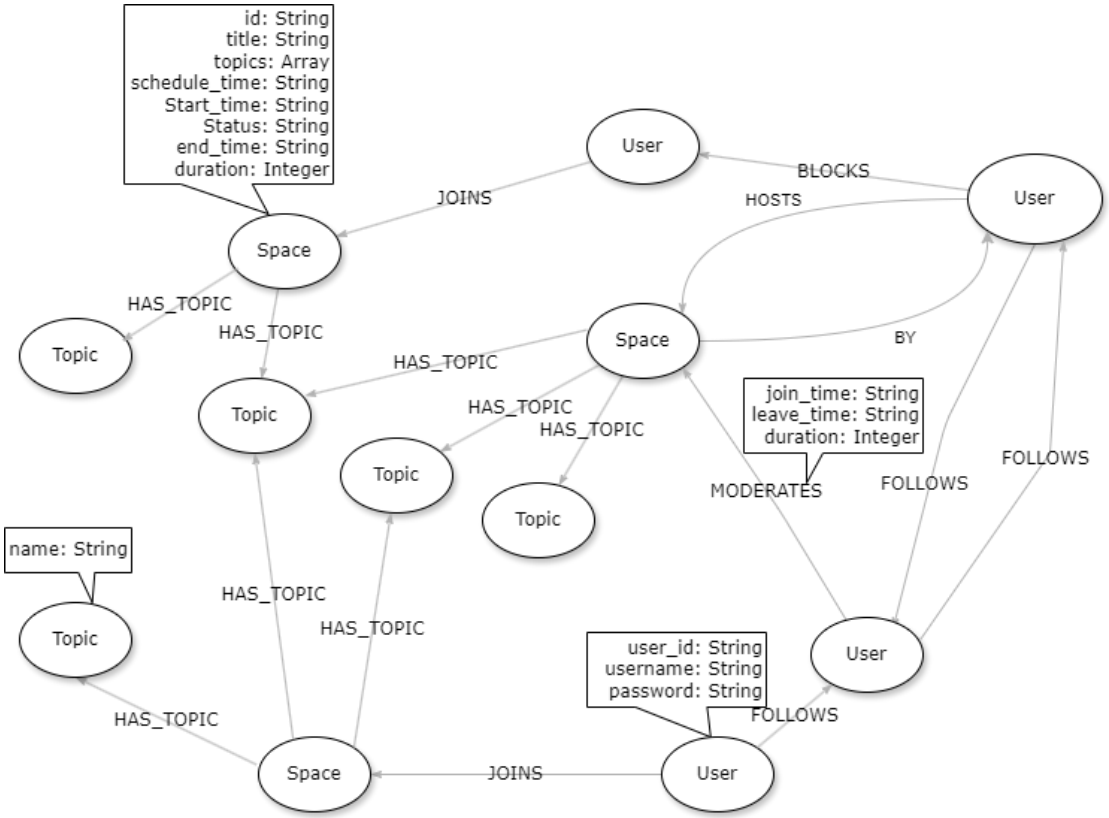

Figure 2: The space data model

3.5 Recommendation Method

In this system, I need to recommend Spaces that users may be interested in. However, these Spaces lack feature information about audio content, so I concentrate on their topics. I choose to calculate the users' interest level in these topics. Based on the level, I calculate user similarities and then recommend Spaces in which similar users participated. I have introduced three similarity coefficients in the background section. Since each Space is limited to a maximum of three topics, and users' preferences for

topics are typically confined to a few areas, users usually associate only a few topics, resulting in a limited number of topics related to any individual user. So, I didn't choose the Jaccard coefficient. In addition, the Pearson similarity is more about judging linear relationships; if I consider users' duration as a weight factor, users and topics or users and users are not necessarily linearly related, so I finally chose cosine similarity. The mathematical calculation method for user similarity is as follows:

From Figure 2, we can find three types of paths between the user and the topic:

User—(HOSTS)→Space—(HAS_TOPIC)→Topic;

User—(MODERATES)→Space—(HAS_TOPIC) →Topic;

User—(JOINS)→Space—(HAS_TOPIC)→Topic.

Each relationship(r) between the user and the Space has a property named duration; I use $r_{duration}$ to represent it. Each space(s) also has a property named duration; I use $s_{duration}$ to represent it. Then, I incorporate $\frac{r_{duration}}{s_{duration}}$ as one of the weighting factors when assessing the user's interest in the Space, and the relationship types should also be taken into consideration; I set the relationship weight as:

$$r_{weight}(i) = \begin{cases} 1.0, & if\ relationship\ is\ JOINS \\ 2.0, & if\ relationship\ is\ MODERATES \\ 3.0, & if\ relationship\ is\ HOSTS \end{cases}$$

Hence, the interest weight of a user in a given Space can be calculated as following formula:

$$r_{\text{weight}}(i) \times \frac{r_{\text{duration}}(i)}{s_{\text{duration}}(i)}$$

There can be different paths between the user and the topic, so the total interest weight of a user in a specific topic can be defined as follows:

$$\text{total\_weight} = \sum_{i=1}^{n} \left(r_{\text{weight}}(i) \times \frac{r_{\text{duration}}(i)}{s_{\text{duration}}(i)} \right)$$

where n represents the number of paths between the user and the topic.

After calculating the interest weight, we can use cosine similarity to calculate

different user's similarity:

$$\text{cosine\_similarity}(V_1, V_2) = \frac{V_1 \cdot V_2}{||V_1|| ||V_2||}$$

and:

$$V_1 \cdot V_2 = \sum_{j=1}^{t} V_1(j) \times V_2(j)$$

$$||V_1|| = \sqrt{\sum_{j=1}^{t} V_1(j)^2}$$

$$||V_2|| = \sqrt{\sum_{j=1}^{t} V_2(j)^2}$$

where *t* represents the number of topics, $V_1 \cdot V_2$ represents the dot product of the weight vectors of two users with respect to a topic, $||V_1||$ and $||V_1||$ represent magnitudes of the two vectors.

Based on the above formula, we can calculate user similarity in view of the User-Topic path. Subsequently, we can choose the top *n* similar users, and recommend to the target user the Spaces in which their most similar users have participated or are currently participating. The recommendation score of a Space can be the sum of the similarity scores of these similar users in this Space.

3.6 Data import and generate

Some of the data I used in this project were feature data captured from Twitter Space without sensitive information, and some were artificial data I created. There are many ways to import the data into the database; in this project, I used Python code to write data into Neo4j. Pandas is an open-source data analysis and manipulation Python library(pandas - Python Data Analysis Library 2024); I can filter and combine the relevant data I need by utilising Pandas DataFrame. Subsequently, the filtered data can be imported into the Neo4j database in a batch.

The Neo4j connection should be established first, as shown in the following figure:

```
graph = Graph(Config.NEO4J_URI, auth=(Config.NEO4J_USER, Config.NEO4J_PASSWORD))
```

Figure 3: Establish the Neo4j connection

The absolute path of the Twitter Space data sample should then be obtained for subsequent processing:

```
current_directory = os.path.dirname(os.path.abspath(__file__))

base_folder_path = os.path.join(current_directory, 'Twitter Spaces Data Sample')

all_space_data = pd.DataFrame()
all_user_data = pd.DataFrame()
all_user_space_data = pd.DataFrame()

for folder_name in os.listdir(base_folder_path):
    folder_path = os.path.join(base_folder_path, folder_name)
```

Figure 4: The code to obtain the path of the data sample

The Python code will be used to open the folder, process the data, and store it in the DataFrame. An example of processing user data is shown in the figure below:

```
    user_space_json_path = os.path.join(folder_path, 'space_user_data.json')
    if os.path.exists(user_space_json_path):
        with open(user_space_json_path, 'r', encoding='utf-8') as file:
            user_json_data = json.load(file)

        # If the JSON file contains multiple records, convert it to DataFrame
        if isinstance(user_json_data, list):
            user_df = pd.DataFrame(user_json_data)
        else:
            user_df = pd.DataFrame([user_json_data])

        user_df = user_df[['space_id', 'user_id', 'username']]

        all_user_data = pd.concat( objs: [all_user_data, user_df[['user_id', 'username']].drop_duplicates()],
                                  ignore_index=True)

        all_user_space_data = pd.concat( objs: [all_user_space_data, user_df], ignore_index=True)

all_user_data = all_user_data.drop_duplicates(subset=['user_id'], keep='first')
```

Figure 5: The code to process the user data

It contains more columns, and some columns need to be adjusted to ensure consistency with the application. The duration should be converted to seconds, as shown in Figure 6, and the start time and end time should be converted to ISO 8601

format, as shown in Figure 7.

```
def convert_duration_to_seconds(duration_str):
    h, m, s = map(int, duration_str.split(':'))
    return h * 3600 + m * 60 + s

if 'duration' in df.columns:
    df['duration_seconds'] = df['duration'].apply(convert_duration_to_seconds)
```

Figure 6: The code for converting the duration to seconds

```
if 'start_date' in df.columns and 'start_time' in df.columns:
    df['start_datetime'] = pd.to_datetime(df['start_date'] + ' ' + df['start_time']).apply(
        lambda x: x.isoformat() + 'Z')

if 'end_date' in df.columns and 'end_time' in df.columns:
    df['end_datetime'] = pd.to_datetime(df['end_date'] + ' ' + df['end_time']).apply(
        lambda x: x.isoformat() + 'Z')
```

Figure 7: The code to format the time

The columns of the all_user_space_data is shown in the following picture:

```
space_data_columns_to_extract = ["id", "title", "scheduled_start", "state", "host_ids", "participant_count",
                                 "creator_id", "total_speakers", "total_moderators", "topic_names",
                                 "duration_seconds", "start_datetime", "end_datetime", "speaker_ids"]
```

Figure 8: The columns of the all_user_space_data

As shown in Figure 9, the relationship will be determined based on the user's role in the Space:

```
def determine_relationship_type(row, host_ids, speaker_ids):
    user_id = str(row['user_id'])
    if isinstance(host_ids, list) and user_id in host_ids:
        return 'HOSTS'
    elif isinstance(speaker_ids, list) and user_id in speaker_ids:
        return 'MODERATES'
    else:
        return 'JOINS'
```

Figure 9: The function to determine the relationship between the user and the Space

In light of the emojis included in the Space titles, I wrote a function to remove

them:

```
def remove_emoji(text):
    if not isinstance(text, str):
        return text

    emoji_pattern = re.compile(pattern="["
                                       "\U0001F600-\U0001F64F"  # Emoticons
                                       "\U0001F300-\U0001F5FF"  # Miscellaneous Symbols and Pictographs
                                       "\U0001F680-\U0001F6FF"  # Transport and Map Symbols
                                       "\U0001F1E0-\U0001F1FF"  # Enclosed Alphanumeric Supplement
                                       "\U00002702-\U000027B0"  # Dingbats
                                       "\U0001F900-\U0001F9FF"  # Supplemental Symbols and Pictographs
                                       "\U0001FA70-\U0001FAFF"  # Symbols and Pictographs Extended-A
                                       "\U0001F1F2-\U0001F1F4"  # Flags (iOS)
                                       "\U0001F1E6-\U0001F1FF"  # Flags
                                       "\U00002600-\U000026FF"  # Miscellaneous Symbols
                                       "\U00002300-\U000023FF"  # Miscellaneous Technical
                                       "\U00002194-\U00002199"  # Arrows
                                       "\U0001F700-\U0001F77F"  # Alchemical Symbols
                                       "\U0001FA70-\U0001FAFF"  # Symbols & Pictographs Extended-A
                                       "\U0001F780-\U0001F7FF"  # Geometric Shapes Extended
                                       "\U0001F800-\U0001F8FF"  # Supplemental Arrows-C
                                       "\U0000203C-\U0000203C"  # Double Exclamation Mark ( ‼ )
                                       "\U00002757-\U00002757"  # Exclamation Mark in a Circle ( ❗ )
                                       "]+", flags=re.UNICODE)
    return emoji_pattern.sub( repl: r'', text)
```

Figure 10: The remove_emoji function

Since there is no data on the length of time that a user has spent in a Space, I created virtual duration properties for the relationships between users and Spaces. When the relationship between the user and the Space is “HOSTS”, the user’s duration in this Space is the Space’s duration; if the relationship is “MODERATES”, the user’s duration in this Space is a random value between half of the Space’s duration and of the Space’s duration, I set this parameter based on the consideration that the time a speaker spends in a Space is generally not significantly too short when compared with the Space's overall duration; if the relationship is “JOINS”, I assigned the user's duration in this Space a random value between 1 and the Space's total duration.

Figure 11 demonstrates the function of generating a random join time, and Figure 12 shows the function of generating the join_time, leave_time, and duration for the relationship between the user and the Space.

```python
def random_time_within_range(start, end):
    delta = end - start
    random_second = random.randint( a: 0, int(delta.total_seconds()))
    return start + timedelta(seconds=random_second)
```

Figure 11: The function for generating random join time

```python
def calculate_relationship_attributes(relationship_type, space_row):
    start_time = pd.to_datetime(space_row['start_datetime'])
    end_time = pd.to_datetime(space_row['end_datetime'])

    if relationship_type == 'HOSTS':
        join_time = start_time.isoformat()
        leave_time = end_time.isoformat()
        duration = space_row['duration_seconds']
    elif relationship_type == 'MODERATES':
        duration = random.randint(space_row['duration_seconds'] // 2, space_row['duration_seconds'])
        max_join_time = end_time - timedelta(seconds=duration)
        join_time = random_time_within_range(start_time, max_join_time).isoformat()
        leave_time = (pd.to_datetime(join_time) + pd.to_timedelta(duration, unit='s')).isoformat()
    else:  # JOINS
        duration = random.randint( a: 1, space_row['duration_seconds'])
        max_join_time = end_time - timedelta(seconds=duration)
        join_time = random_time_within_range(start_time, max_join_time).isoformat()
        leave_time = (pd.to_datetime(join_time) + pd.to_timedelta(duration, unit='s')).isoformat()

    return join_time, leave_time, duration
```

Figure 12: The function for generating duration for the user-Space relationships

Next, we can create a DataFrame to store the relationships between users and spaces:

```python
user_space_relationships = []

for _, space_row in all_space_data.iterrows():
    space_id = space_row['id']
    host_ids = space_row.get( key: 'host_ids', default: [])
    speaker_ids = space_row.get( key: 'speaker_ids', default: [])
    creator_id = space_row['creator_id']

    for _, user_row in all_user_space_data.iterrows():
        if user_row['space_id'] == space_id:
            relationship_type = determine_relationship_type(user_row, host_ids, speaker_ids)
            join_time, leave_time, duration = calculate_relationship_attributes(relationship_type, space_row)
            user_space_relationships.append({
                "space_id": space_id,
                "user_id": user_row['user_id'],
                "username": user_row['username'],
                "relationship_type": relationship_type,
                "join_time": join_time,
                "leave_time": leave_time,
                "duration": duration
            })

user_space_relationships_df = pd.DataFrame(user_space_relationships)
```

Figure 13: The code to store the relationship properties in a DataFrame

Finally, we can import this data in batches. Figures 14 and 15 illustrate an example of importing user data, which is similar to the process for importing space and topic nodes, as well as other relationships, into the Neo4j database.

```
def import_user_data(tx, user_id, username, password):
    user = Node( *labels: "User", username=username, password=password, user_id=user_id)
    tx.merge(user, "User", "user_id")
```

Figure 14: The import_user_data function

```
def batch_import_user_data(dataframe):
    default_password = "123"
    hashed_password = bcrypt.hashpw(default_password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

    tx = graph.begin()
    try:
        for _, row in dataframe.iterrows():
            import_user_data(tx, row['user_id'], row['username'], hashed_password)
        graph.commit(tx)
    except Exception as e:
        graph.rollback(tx)
        print(f"Error importing user data: {e}")
```

Figure 15: The batch_import_user_data function

The result of importing the data into the Neo4j database is as follows:

Figure 16: The imported data in Neo4j

The nodes whose colour is green represent Spaces, the blue nodes represent users, and the pink nodes represent topics. From the figure above, we can get the information that most users only participated in one Space, so the recommended results for these users may not be accurate; some manual data needs to be manually added when testing the recommendation engine. The topics of a Space should be reasonable and represent the features of the Space, so it does not seem to be a reasonable choice to generate topics for Spaces and assign virtual Spaces to users randomly. Therefore, I did not choose to create artificial Space data by code.

4. System Design

After analysing the functional requirements, I can determine the main function of the system. Below is the system structure I designed:

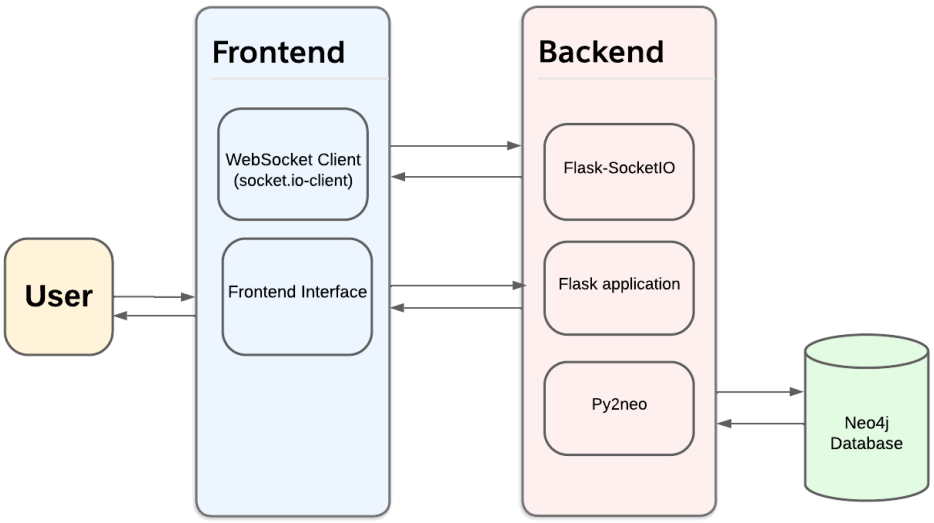

Figure 17: The system architecture

The client will send and receive a real-time data stream from the server through WebSocket. On the other hand, HTTP requests are also used for non-real-time data

communication. The backend uses Py2neo to connect to the Neo4j database.

In terms of the system architecture, I will introduce the frontend design and the folder structure establishment.

4.1 User interface

Since the user needs to view recommended results in this system, there should be a webpage for users which provides personalised recommended Spaces. The home page will display all the Spaces in the database, which can be compared with the recommendation page. In addition, the system should at least include a webpage for users to create Spaces and a webpage of each Space's details to simulate the basic functions of Twitter Space.

The user interface should be clear, intuitive, and with easily accessible functionalities. Therefore, I designed a top navigation bar and a side navigation bar for each web application page. Which is shown in the following picture:

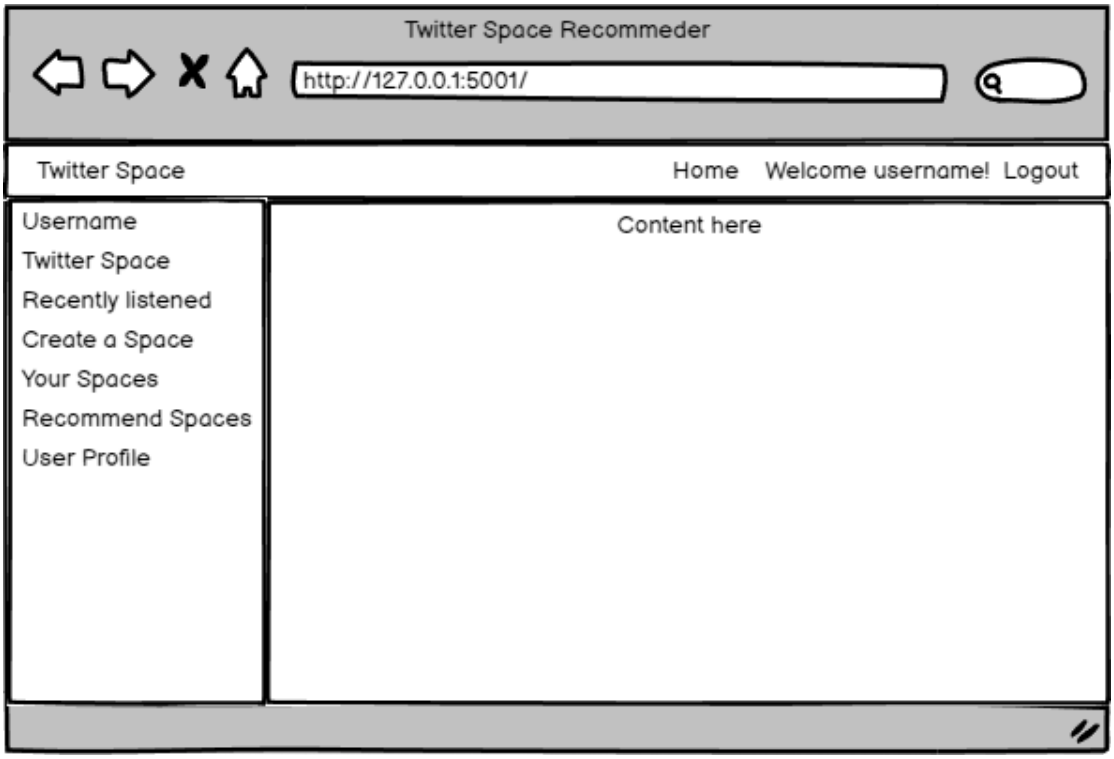

Figure 18: The basic webpage design

The home page displays all the Spaces in the database. Every live Space has a "Join Space" and a "Moderate Space" button, and every ended Space has a "Space Ended" button to view the participants' details of this Space. For upcoming Spaces, users can view the expected start time and other basic information about them.

The recommended Spaces page is similar to the home page, but it displays the Spaces recommended for the current user.

There is a page for users to view all the Spaces they have created. On this page, the user can start a Space with an upcoming status or return to a Space they initiated but did not conclude.

When the user clicks the "Join Space" or the "Moderate Space" button, or the user starts the Space, the space details page will be demonstrated, and there is a "End Space" button for the Space creator and a "Leave Space" button for other participants.

The create a Space webpage needs to include a form that allows the user to enter the Space title, choose the expected start time of the Space and choose the topics of the Space. There are many topics to choose from, so the topic selection is presented to the user in the form of a modal dialogue; users are provided with several topic options to choose from, and if the pre-defined topic options can't meet the needs, users can input and customise their own topics.

4.2 Folder Structure

In order to better manage my code, I designed the folder structure at the beginning of this project, which is shown in the following picture:

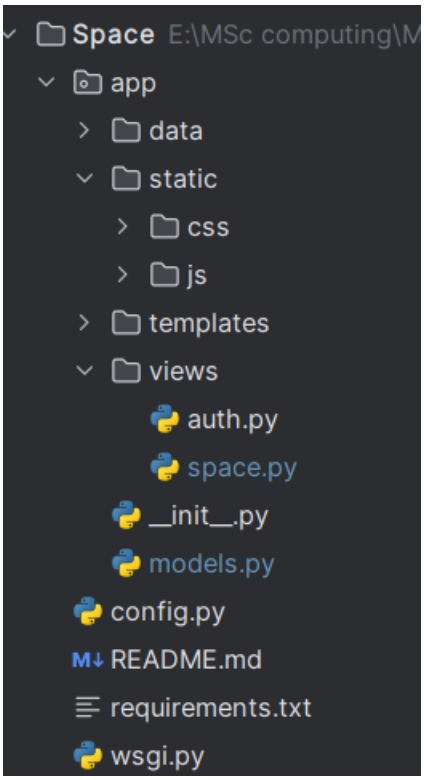

Figure 19: The folder structure

The config.py mainly includes the database configuration and the secret key and enables the APScheduler. The models.py stores the code that interacts with the database, and the views folder stores the code that processes the client's requests, calls functions from the models.py, and responds to the client. I use the blueprint to manage user authentication and main Space function modules; this helps me keep the code files from becoming too bloated.

5. Implementation

5.1 User Authentication

The first function I implemented is about user authentication. There are two solutions to implement this function; one is to allow the user who has not logged in to access some pages of this system and remind users to log in before accessing some pages; another solution is to restrict users who have not logged in from accessing all

the webpages of this system. Since my initial plan was that the home webpage would display recommendation content, I chose the second option when I started implementing this function. Some applications require additional information during registration, such as email, interests, etc. However, the primary purpose of this project is to simulate the recommendation process in audio-based social networks, so the system only requires the username and password information without more verification functions.

I created a “User” class to add some functions that users can perform on the system. This class has functions for creating users and verifying passwords. The create_user function takes the current object instance(‘self’) and a ‘password’ parameter as input; when the current object instance has no matching data in the database, the password parameter inputted will be encrypted by Bcrypt and then the function creates a new node in the database, The username of the node is the username of the current instance, the password of the node is the encrypted password, the user_id of the node is the id automatically generated by the system. The verify_password function also accepts ‘self’ and a ‘password’ parameter; after confirming that the user exists in this system, the Bcrypt will be used to check the ‘password’ parameter matches the user's password. These functions are then utilised in the auth.py module, and Flash is used to give appropriate reminders when errors occur.

The final user login and registration page is shown in the figure below:

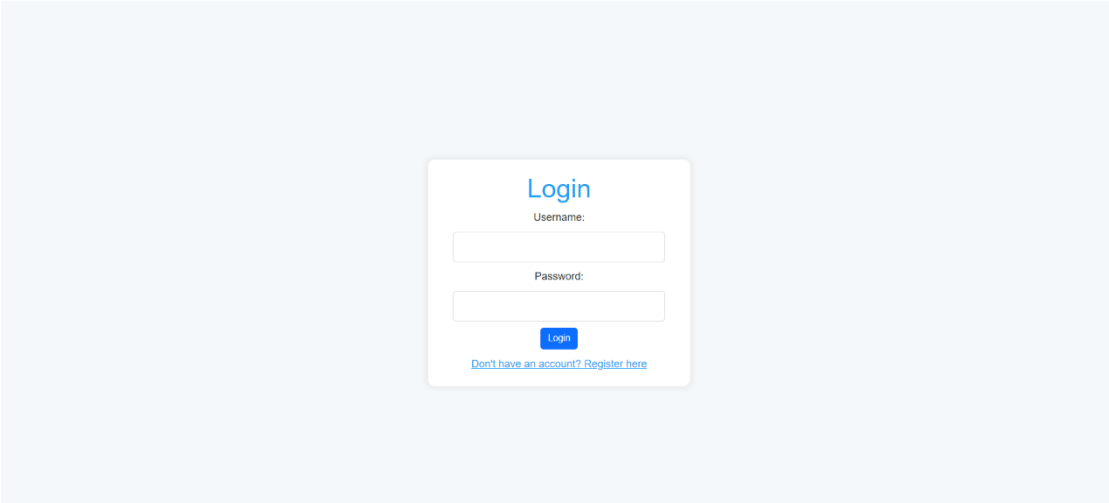

Figure 20: The login page

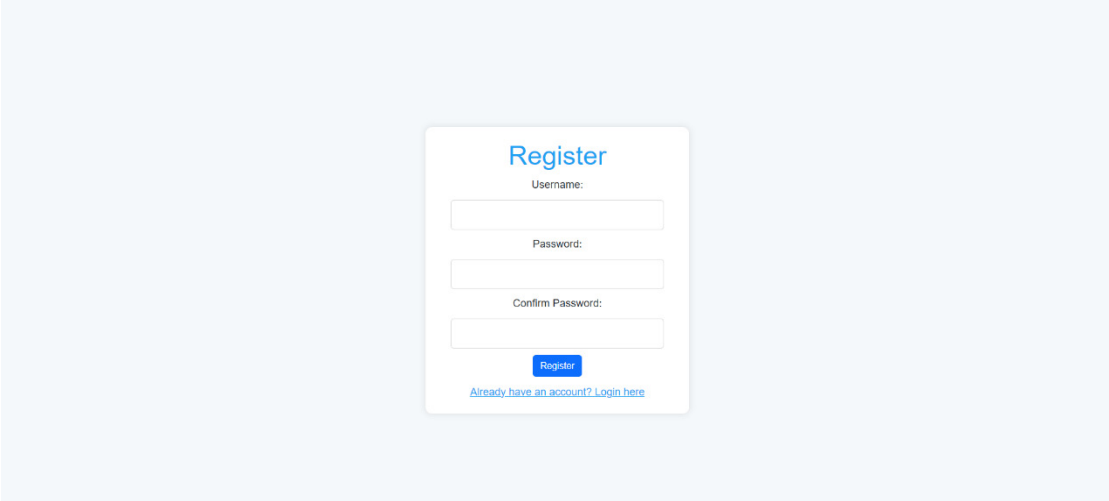

Figure 21: The registration page

5.2 Create a Space Function

When I first designed the Create a Space function, I decided to build a complete topic selection box like Twitter Space so that users do not need to enter the topics themselves. However, there were too many topics, so I finally created part of the topic options and allowed users to add topics by themselves. The topic selection part is mainly implemented on the front end. The HTML code of the topic selection is shown in the following picture:

```
<div class="mb-3 topics">
    <label class="form-label">Topics</label>
    <div class="mb-2 topic-buttons" id="selected-topics">
        <!-- Dynamically generated topic buttons -->
    </div>
    <button type="button" class="btn btn-secondary add-topic-button w-100">Add Topics</button>
</div>

<!-- Hidden input to store selected topics -->
<input type="hidden" id="selected-topics-input" name="selected_topics" value="">
```

Figure 22: The HTML code for topic selection

The selected topics are stored in a hidden input element, which will be subsequently transmitted to the back end. The topic buttons are displayed dynamically through the generation of these buttons using JavaScript.

A topic selection modal is provided for users to add topics. Each topic button is assigned the class “topic-button”, the modal includes a save button with the CSS class “save-button” that allows users to save their selected topics upon each interaction. The user interface utilises the 'selected' CSS class to differentiate between selected and unselected topics, thereby enhancing the intuitiveness of user interactions. When a user clicks a topic button, if the button is already selected, the 'selected' CSS class is removed. If the button is not selected, the system checks whether the user has selected three topics; if fewer than three topics are selected, the 'selected' class is added to the button element.

Additionally, the user can add their own topics by entering the topic content and then clicking the “Add your Topic” button. If fewer than three topics have been selected, a button element, consistent in CSS class with the other topic buttons in the modal, will be automatically generated and selected by default.

When the “Save” button in the topic selection modal is clicked, the div element with the id “selected-topics” will be updated, and the hidden input will be updated as the selected topics button in the div element updates.

The variables defined in the “Save” button related function are shown in the figure below:

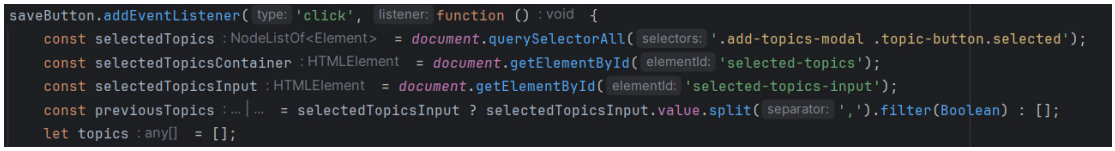

Figure 23: The variables defined in the "Save" button function

The figure below demonstrates the JavaScript code to handle the selected topics sent to the back end:

```
selectedTopics.forEach( callbackfn: topic : Element => {
    if (!topics.includes(topic.textContent)) {
        topics.push(topic.textContent);
    }
});

if (selectedTopicsContainer && selectedTopicsInput) {
    if (topics.join(',') !== previousTopics.join(',') && topics.length <= 3) {
        selectedTopicsContainer.innerHTML = '';
        topics.forEach(topic => {
            const newButton : HTMLButtonElement = document.createElement( tagName: 'button');
            newButton.type = 'button';
            newButton.className = 'topic-button selected';
            newButton.textContent = topic;
            newButton.addEventListener( type: 'click', listener: function () : void {
                newButton.remove();
                topics = topics.filter(t => t !== topic);
                selectedTopicsInput.value = topics.join(',');
            });
            selectedTopicsContainer.appendChild(newButton);
        });
        selectedTopicsInput.value = topics.join(',');
    }
}
```

Figure 24: The JavaScript code to generate the topic buttons and inputs

In the back end, there is the create_space function, which accepts several parameters from the front end, including the title, topics list and schedule_time. The function uses uuid to generate a Space ID number, and the schedule_time is converted to UTC for future extension. The Space node is created by using database transactions, and the "merge" method of py2neo is used to check if there are corresponding nodes in the database before creating new topic nodes. The "HOSTS" and "BY" relationships between the user and the Space and the "HAS_TOPIC" relationships from the Space to

the topic have also been created by using database transactions.

The Create a Space webpage is shown in the figure below:

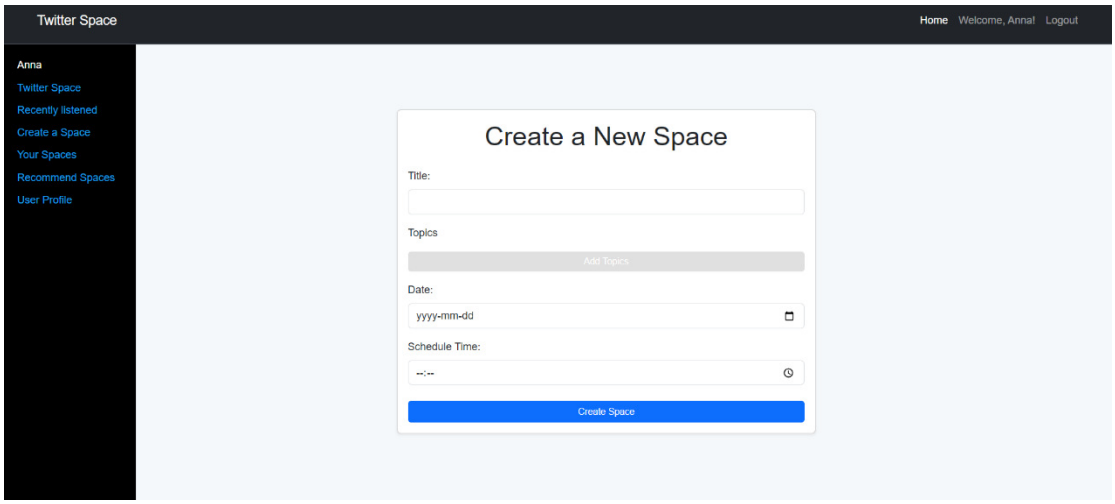

Figure 25: The Create a Space webpage

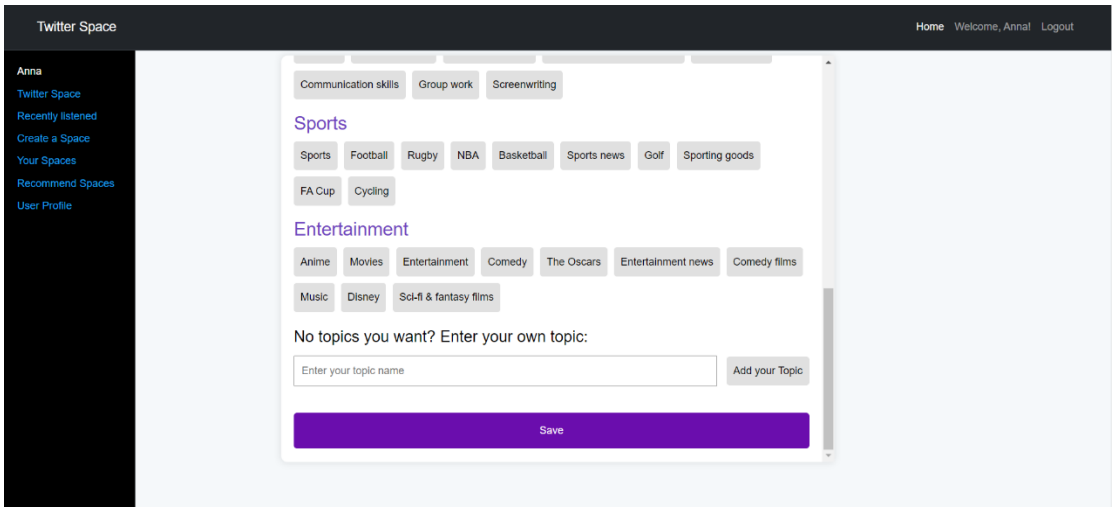

Figure 26: The topic selection modal

5.3 Start a Space Function

In order to make sure that the Space creator can receive a notification when the expected start time arrives, I wrote a check_schedule_time function; this function would retrieve the current time and then search the Spaces whose status is “upcoming” and schedule_time is earlier than the current time, the Cypher code is as the following figure:

```
MATCH (host:User)-[:HOSTS]->(s:Space {status: 'upcoming'})
WHERE s.schedule_time <= $current_time and s.start_time IS NULL
RETURN s.id AS space_id, host.username AS host_username
```

Figure 27: The Cypher query to find upcoming Spaces and their hosts

APScheduler is used to call the check_schedule_time function regularly; when the function has results, Flask-SocketIO send a message to the front-end for every Space in the results; and then the front-end determines whether the current user is the Space creator after receiving the messages. The code to invoke the function at regular intervals is demonstrated in Figure 28:

```
@scheduler.task('interval', id='do_check_schedule_time', seconds=20, misfire_grace_time=900)
def remind_schedule_time():
    logging.info('Function to remind that the scheduled time started')
    try:
        from .models import check_schedule_time
        spaces = check_schedule_time()
        for space in spaces:
            socketio.emit( event: 'schedule_reminder', *args: {'space_id': space['space_id'],
                                                'host_username': space['host_username']}, room=None)
        logging.info('Function to remind that the scheduled time executed successfully')
    except Exception as e:
        logging.error(f'Error executing function to remind the scheduled time: {e}')
```

Figure 28: The code to call the check_schedule_time function regularly

If the current user is the creator of a Space in the results, a notification will be displayed on the page, as is shown in the following figure:

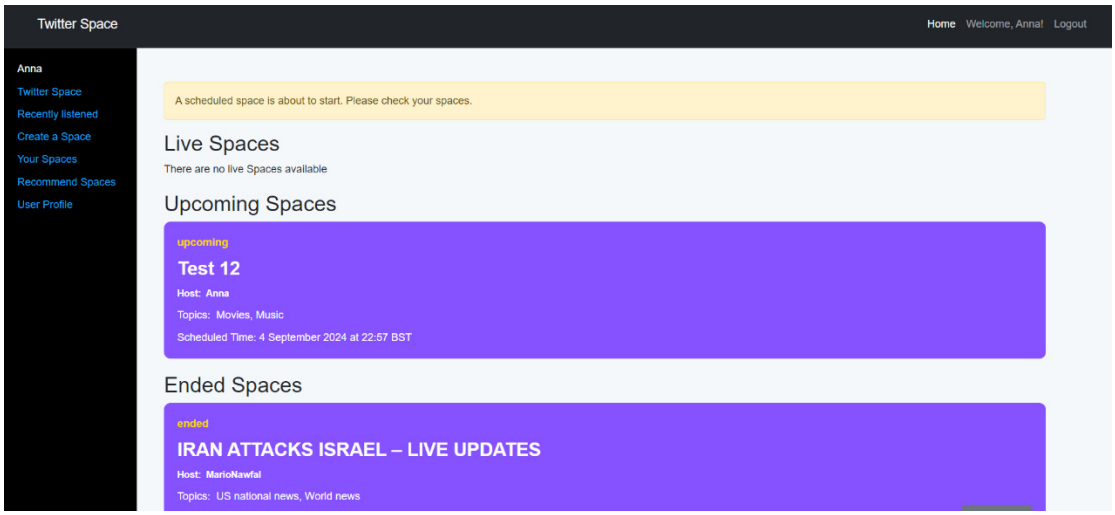

Figure 29: The displayed notification

The start_space function is inside the User class; it takes the ‘self’ and ‘space_id’ as parameters. The host will generally join the Space automatically when they start a Space, so the operation which determines the current user is in other Spaces is required. The Cypher query is shown in the following figure:

```
MATCH (u:User {username: $username})-[r]->(s:Space {status: 'live'})
WHERE TYPE(r) IN ['JOINS', 'MODERATES', 'HOSTS']
    AND (s.start_time IS NULL OR s.start_time <= $current_time)
    AND s.end_time IS NULL
    AND r.leave_time IS NULL
    AND r.join_time IS NOT NULL
RETURN s
```

Figure 30: The Cypher query to check if the current user is in another Space

Once it is confirmed that the current user is not in another live Space, the status and start_time of the Space will be updated, and the join_time of the “HOSTS” relationship will also be updated:

```
MATCH (u:User {username: $username})-[r:HOSTS]->(s:Space {id: $space_id})
SET s.status = 'live', s.start_time = $current_time, r.join_time = $current_time
RETURN s
```

Figure 31: The Cypher statement to start a Space

5.4 End a Space Function

The end_space function is also in the User class; this function also accepts the “self” and “space_id” as parameters. Firstly, this function checks if the current user is the Space creator; if the user is the creator, then the Space status will be updated from “live” to “ended”, and the duration and leave_time of users who are currently in the Space will also be updated. Whether the leave_time is null is the condition of judging whether the user is in the Space; if it is null, the user is considered to be in the Space. The Cypher statement is shown in the figure below:

```
MATCH (s:Space {id: $space_id})
WHERE s.status = 'live'
SET s.end_time = $current_time, s.status = 'ended',
    s.duration = duration.between(datetime(s.start_time), datetime($current_time)).seconds
WITH s
MATCH (u:User)-[r]->(s)
WHERE TYPE(r) IN ['JOINS', 'MODERATES', 'HOSTS'] AND r.leave_time IS NULL
WITH s, u, r, duration.between(datetime(r.join_time), datetime($current_time)).seconds as session_duration
SET r.duration = coalesce(r.duration, 0) + session_duration,
    r.leave_time = $current_time
RETURN s, u, r
```

Figure 32: The Cypher statement for the end_Space function

In space.py, the socket.io is used to broadcast a message that the Space ends after calling the end_space function, and the Space status in the user interface will be updated promptly. The user interface is shown in the figure below:

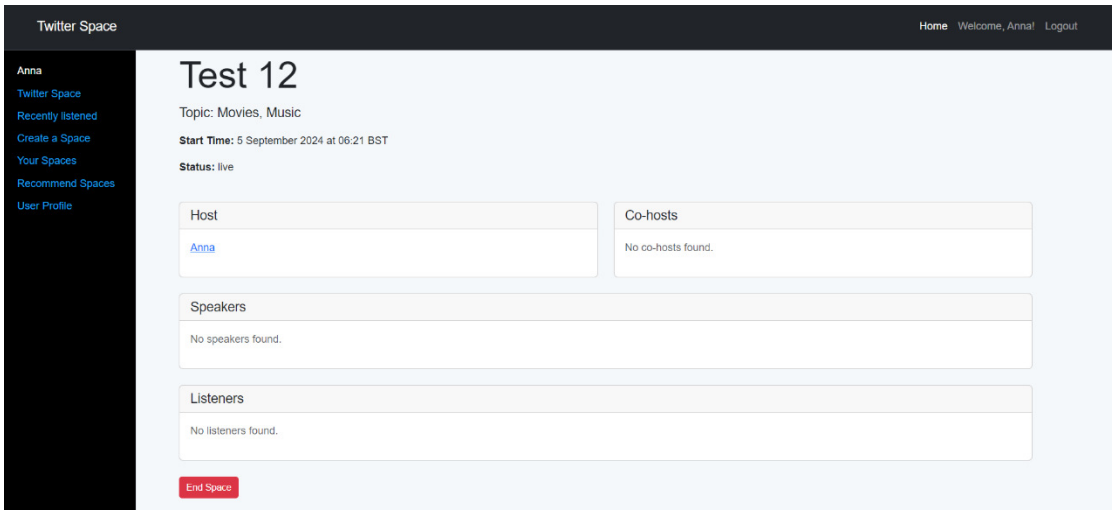

Figure 33: The Space webpage

5.5 Join and Moderate a Space Function

I encountered a challenge when I wrote functions for this part. At first, I simply check whether the user is in another Space by using the Cypher query in Figure 30; if not, then create a “JOINS” or “MODERATES” relationship. However, I found that in this case, the situation in which the user rejoins the Space cannot be handled correctly. If the user joins the same Space multiple times, their duration can’t be accumulated correctly. Then, I considered two solutions. The first solution is to create a relationship

every time the user joins or moderates the Space, and the total duration a user spends in a space will be calculated by summing the durations of all relationships between the user and the space. Although this method can retain the user's historical interaction records for the Space, it may result in too many relationships from a user to a Space, making it more complicated to calculate the user's weight toward the topic. Also, the time complexity will be higher. The second solution is to use 'merge' to create or update the relationships when the user joins a Space. As shown in Figure 34, the join_time is set as the current time; the leave_time is set to null; when the user leaves this Space, the leave_time will be updated, and the duration will be accumulated. Since my recommendation method concentrates on the total duration and a complete user history interaction record is unnecessary, I finally chose the second solution for performance considerations.

```
MATCH (u:User {username: $username})-[r:JOINS]->(s:Space {id: $space_id})
SET r.leave_time = NULL
```

Figure 34: The Cypher statement for joining a Space

5.6 Leave a Space Function

This function is related to joining or moderating a Space function. As mentioned before, I chose to reset the leave_time to null when the user rejoins a Space, so we can infer that the user is in the Space if the user's join_time is not null and leave_time is null. Therefore, the leave_space function firstly determines whether the user has hosted the Space; if not, it can continue to execute the following queries:

```
MATCH (u:User {username: $username})-[r]->(s:Space {id: $space_id})
WHERE TYPE(r) IN ['JOINS', 'MODERATES'] AND r.leave_time IS NULL AND r.join_time IS NOT NULL
WITH u, r, duration.between(datetime(r.join_time), datetime($leave_time)).seconds as session_duration
SET r.duration = coalesce(r.duration, 0) + session_duration,
    r.leave_time = $leave_time
RETURN u
```

Figure 35: The Cypher statement for leaving a Space

5.7 Social Interaction Function

The principals of the follow, unfollow, block, and unblock functions are similar; these functions all accept “self” and another user’s “username” as parameters. The follow and block functions need to determine whether the user already has a block or follow relationship with another user after confirming that another user exists and the current user isn’t equal to another user. In theory, a user will not follow and block another user simultaneously, so the follow and block relationships from one user to another are incompatible. For example, to create a “BLOCKS” relationship from user 1 to user 2, tx.seperate() will be used to delete the “FOLLOWS” relationship from user 1 to user 2 at first. The Unfollow and Unblock functions are slightly simpler than the follow and block functions, with only two operations: checking whether there is a “FOLLOWS” or “BLOCKS” relationship and then deleting this relationship.

The get_social_relationships function will search for relationships between this user and other users and then store these relationships in a dictionary. The final user profile page is as follows:

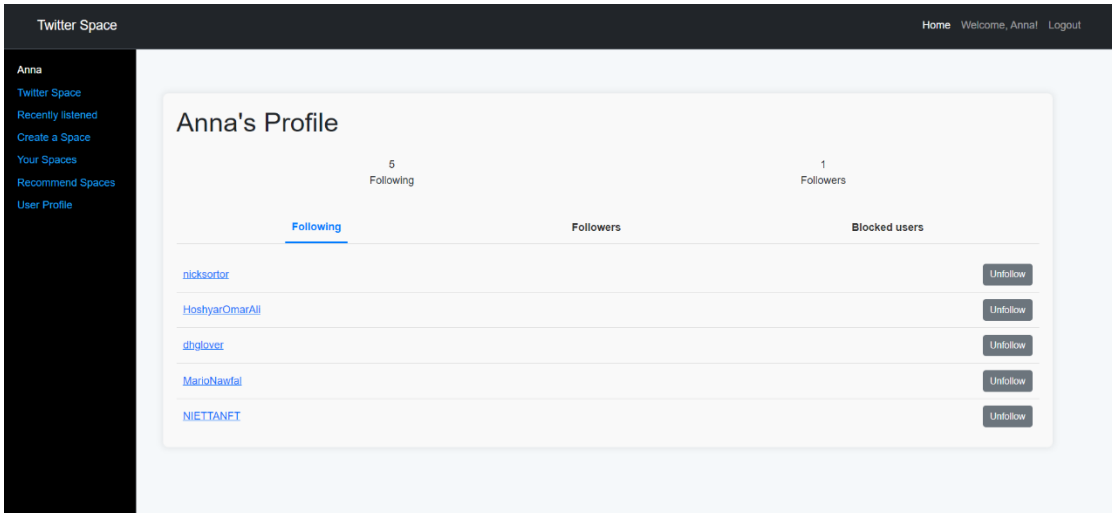

Figure 36: The webpage for viewing a user's own profile

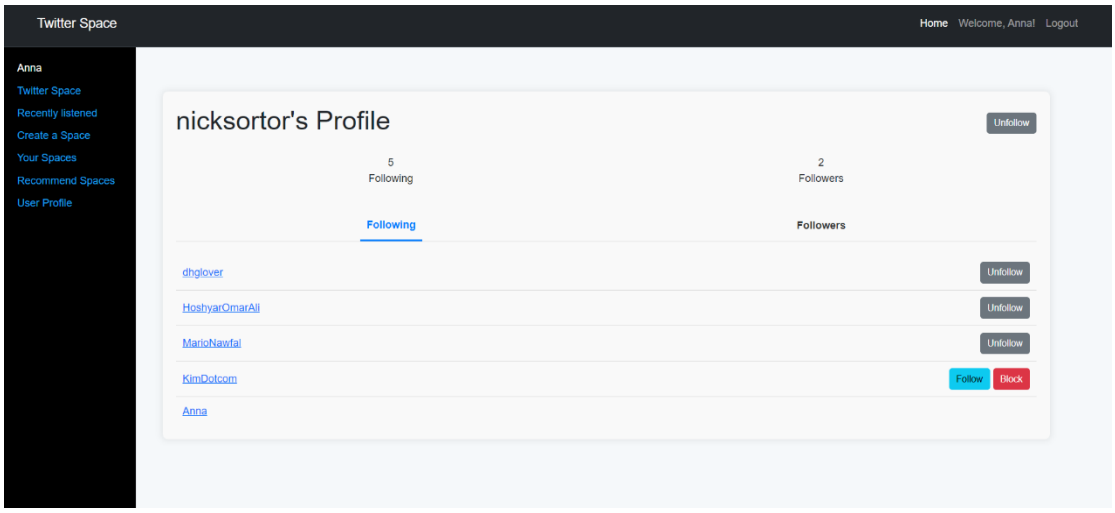

Figure 37: The webpage for viewing another user's profile

5.8 Recommend Spaces

As discussed in Section 3.5, the recommendation function should be implemented by following the steps outlined in the algorithm. I used Cypher to calculate user similarity at first, but it was a bit complicated for me to implement it with Cypher's statement; the statement was too long, and it was easy to make mistakes and hard to find where the error occurred. Therefore, I finally used Cypher's statement to get the user's interested topic weight and then used some Python libraries to compute the cosine similarity. As shown in the following figure, the user's total interest weight on a topic was calculated using a Cypher statement, and then Pandas was used to store the data as a DataFrame structure.

```
def get_user_topic_weights_df():
    query = """
    MATCH (u:User)-[r]->(s:Space)-[:HAS_TOPIC]->(t:Topic)
    WITH u.username AS user, t.name AS topic,
         CASE
             WHEN type(r) = 'JOINS' THEN 1.0 * r.duration / s.duration
             WHEN type(r) = 'MODERATES' THEN 2.0 * r.duration / s.duration
             WHEN type(r) = 'HOSTS' THEN 3.0 * r.duration / s.duration
             ELSE 0
         END AS weight
    WHERE weight > 0
    RETURN user, topic, SUM(weight) AS total_weight
    """
    data = graph.run(query).to_data_frame()
    return data
```

Figure 38: The function to calculate the user's interest topic weight

Considering that the utility matrix is likely to be a sparse matrix, I use the coo_matrix from scipy.sparse to construct the user-topic weight matrix. The function is demonstrated in the following figure.

```
def build_sparse_matrix(data):
    users = data['user'].unique()
    topics = data['topic'].unique()

    user_index = {user: i for i, user in enumerate(users)}
    topic_index = {topic: j for j, topic in enumerate(topics)}

    rows = data['user'].map(user_index)
    cols = data['topic'].map(topic_index)
    weights = data['total_weight']

    sparse_matrix = coo_matrix( arg1: (weights, (rows, cols)), shape=(len(users), len(topics)))

    return sparse_matrix, users, topics
```

Figure 39: The function to build the user-topic weight matrix

Then, I use the cosine_similarity function from the sklearn.metrics.pairwise module to calculate user similarity. The following picture illustrates the function of calculating the cosine similarity.

```
def compute_similarity(sparse_matrix):
    similarity = cosine_similarity(sparse_matrix, dense_output=False)
    return similarity
```

Figure 40: The function to calculate user similarity

Since this algorithm is invalid for new users or users without Space interaction records, a get_default_recommendations function was created to recommend the most popular Spaces; the Cypher statement is shown in the figure below:

```
MATCH (s:Space)
OPTIONAL MATCH (u:User)-[r:JOINS|MODERATES|HOSTS]->(s)
WHERE (s.status = 'live' AND r.leave_time IS NULL) OR (s.status = 'ended')
WITH s, COUNT(DISTINCT u) AS participant_count
RETURN s.id AS id, s.title AS title, s.topics AS topics, s.start_time AS start_time,
       s.schedule_time AS schedule_time, s.end_time AS end_time, s.status AS status,
       participant_count
ORDER BY participant_count DESC
LIMIT 10
```

Figure 41: The Cypher statement for finding the most popular Spaces

When the user has participation records, the Spaces that the top 20 most similar users have participated in will be recommended, and the sum of the similarities of similar users in the Space will be used as the participation score, as shown in the figure below:

```
MATCH (u1:User {username: $username})-[:JOINS|MODERATES|HOSTS]->(s1:Space)
WITH COLLECT(s1) AS user_spaces
MATCH (u2:User)-[r:JOINS|MODERATES|HOSTS]->(s:Space)
WHERE u2.username IN $similar_usernames AND NOT s IN user_spaces
OPTIONAL MATCH (s)-[:BY]->(h:User)
OPTIONAL MATCH (u:User)-[r2:JOINS|MODERATES|HOSTS]->(s)
WHERE (s.status = 'live' AND r2.leave_time IS NULL) OR (s.status = 'ended')
WITH s, h, SUM($similarity_map[u2.username]) AS weighted_participation,
     COUNT(DISTINCT u) AS participant_count, COLLECT(DISTINCT u.username) AS participants
RETURN s.id as id, s.title as title, s.topics as topics, s.start_time as start_time,
       s.schedule_time as schedule_time, s.end_time as end_time, s.status as status,
       h.username as host_name, weighted_participation, participant_count, participants
ORDER BY weighted_participation DESC, participant_count DESC
LIMIT 10
```

Figure 42: The Cypher statement for recommendation

6. Testing and Results

The testing includes basic function testing and recommendation engine testing;

they will be introduced in different ways.

6.1 Test Cases

The basic function is tested by following the test cases; the following tables are the test cases I created.

Test Case Id: 01 (Log in function)	**Test Purpose:** Verify requirement – users can log in to an account successfully.		
Environment: Google Chrome; Windows 11			
Preconditions: The user has an account in the database.			
Test Case Steps: Basic flow			
Step No	Procedure	Expected Response	Pass/Fail
1	Access the website	The website navigates to the login page and displays the username and password text box and a registration link.	pass
2	Enter a username which exists in the database and the correct password, and click the “Login” button.	The website redirects to the home page.	pass
3	Enter a username which exists in the database and a wrong password, and click the “Login” button.	The website displays a notification to the user and remains on the login page	pass
4	Enter a username which does not exist in the database and a password, and then click the “Login” button.	The website displays a message to remind the user that the user does not exist and redirects to the register page.	pass
5	Click the “Login” button without entering the username or password.	The form does not submit to the back end, and the website	pass

		displays a message reminding the user to fill in both the username and password fields.	
Comments: This test case is related to the register and logout function test case. It does not handle situations where users forget their passwords.			

Table 3: The login function testing

Test Case Id: 02 (Register function)	**Test Purpose:** Verify requirement – users can register an account successfully.		
Environment: Google Chrome; Windows 11			
Preconditions: The user has no account in the database.			
Test Case Steps: Basic flow			
Step No	Procedure	Expected Response	Pass/Fail
1	Access the website and click the register link.	The website displays a register form which includes the “username”, “password”, and “confirm password” input fields.	pass
2	Enter a username that exists in the database, fill in the “password” and “confirm password” input fields, and click the “Register” button.	The website displays a message to remind that the user has already registered before and stays on the register page.	pass
3	Enter a new username and fill in the 'password' field, ensuring it matches the 'confirm password' field.	Create a new user node in the neo4j database, and the website redirects to the login page.	pass
4	Enter a new username and fill in the 'password' field, but the 'confirm	The website displays a message to remind the user that passwords	pass

	password' field does not match the 'password' field and click the "Register" button.	do not match and remains on the login page.	
5	Click the "Register" button without filling in all the input fields.	The form is not submitted to the back end, and the website displays a message to remind the user to fill it out.	pass
Comments: This test case is related to the log-in and log-out function test case. The registration function does not require an email address and imposes no restrictions on the use of special characters or the password length.			

Table 4: Registration function testing

Test Case Id: 03 (Log out function)	**Test Purpose:** Verify requirement – users can log out of an account successfully.		
Environment: Google Chrome; Windows 11			
Preconditions: The user has already logged in to this system.			
Test Case Steps: Basic flow			
Step No	Procedure	Expected Response	Pass/Fail
1	Click the "Logout" button in the navigation bar.	Delete the user's session, log the user out of the system, and redirect to the login page.	pass
Comments: This test case is related to the login and registration function test case.			

Table 5: Log out function testing

Test Case Id: 04 (Create, start and	**Test Purpose:** Verify requirement – The host can create an audio chat room and start this room; after starting the room, they can end the room.

end an audio chat room)			
Environment: Google Chrome; Windows 11			
Preconditions: The host has logged in to this system.			
Test Case Steps: Basic flow			
Step No	Procedure	Expected Response	Pass/Fail
1	Click the “Create a Space” link.	The website displays a “Create a New Space” form; the host can enter the Space title, choose the topics and the start date and time.	pass
2	Click the “Add Topics” button	A modal appears in the middle of the screen, including multiple topic buttons for the host to choose; a text box below the topic buttons for the host to add their own topics.	pass
3	Choose (or enter) up to three topics and click the “Save” button	The chosen topics are displayed in the form.	pass
4	Click the “Add Topics” button, click the selected topic button again and click the “Save” button	The topic buttons which have been clicked again disappear from the form.	pass
5	Click the “ Add Topics” button, deselect the previously selected topic, click other topic buttons (no more than three) and click the “Save” button.	The selected topics displayed in the form are updated.	pass

6	Click the “Add Topics” button, select the fourth topic, and then click the “Save” button.	The message box appears at the top of the page, reminding the host that each Space allows a maximum of three topics.	pass
7	Click the calendar icon in the date input field.	A date selection box is shown, and the hosts can select today and a further date.	pass
8	Click the “Create Space” button without entering a title and choose the start date and time.	A message is displayed to remind the host that the title, date and time should not be blank.	pass
9	Fill in all the input boxes, choose a time before the current time, and click the “Create Space” button.	A message is displayed to remind that the set scheduled time should be a further time.	pass
10	Choose a date from today onwards, click the clock icon in the schedule time input field to choose a subsequent time, fill in all the input boxes and click the “Create Space” button.	A Space node has been created in the database; topic nodes are created(if the Space has new topics); a “HOSTS” and “BY” relationship is created between the creator and the Space; the “HAS_TOPIC” relationships is created between the Space and the Topic nodes, the Space includes title, topics, schedule_time properties that the creator chose.	pass
11	After Creating a Space successfully, click the “Your Spaces” button.	The web page displays all Spaces the user created, and the newly created Space is displayed at the	pass

		top of the list with a “Start Early” button.	
12	The set schedule time arrives	A notification is displayed on the top of the content to remind the host to start the Space.	pass
13	After the scheduled time has arrived, click on the 'Your Spaces' button.	The button of the newly created Space changed to “Start Space”.	pass
14	Click the “Start Space” button.	The website redirects to the Space page and displays the information about Space; the database updates the start time of Space.	pass
15	Other users participated in this Space.	The Space page updates users in the Space without Manual refresh.	pass
16	The host browses other pages of this system and then returns to the “Your Space” page.	The live Space is displayed first, and the button is converted into “Space is Live”; the host can return to the Space page by clicking this button.	pass
17	The host clicks the “End Space” button on the Space page.	The Space end time and duration are updated, and relationships between users and this Space are updated.	pass
Comments: This test case is related to other users’ join and leave functions.			

Table 6: Create, start and end an audio chat room function testing

Test Case Id: 05 (Join, moderate and leave an audio chat room)	**Test Purpose:** Verify requirement – The user can join or moderate a live audio chat room and leave the room before the room ends, and they can rejoin and leave the room multiple times as long as the room has not ended.		
Environment: Google Chrome; Windows 11			
Preconditions: The user has logged into the system, and a live audio chat room is available.			
Test Case Steps: Basic flow			
Step No	Procedure	Expected Response	Pass/Fail
1	Click the “Join Space” button.	The website redirects to the Space page, which shows the host, co-hosts, speakers, and listeners in the Space and other Space information. The “JOINS” relationship is created between the user and the Space in the database, and the join_time property of the relationship is set.	pass
2	Click the “Leave Space” button	The website redirects to the home page, and the relationship's leave_time and duration properties are set in the database.	pass
3	Click the “Moderates Space” button of the previous Space.	The website redirects to the Space page; the “MODERATES” relationship is created in the database with the join_time property.	pass

4	Click the "Leave Space" button.	The website redirects to the home page, and the "MODERATES" relationship's leave_time and duration properties are set in the database.	pass
5	Click the "Join Space" button of the previous Space.	The website redirects to the Space page, and the properties of the "JOINS" relationship is updated, the join_time is set equal to the current time, the leave_time is deleted, the duration doesn't change.	pass
6	Click the "Leave Space" button.	The website redirects to the home page, and the duration and leave_time are updated in the database.	pass
Comments: This test case is related to the creating, starting and ending audio chat room functions.			

Table 7: Join, moderate and leave an audio chat room function testing

Test Case Id: 06 (Follow, unfollow, block and unblock other users)	**Test Purpose:** Verify requirement – The user can follow or block other users and unfollow/unblock the users they followed/blocked.
Environment: Google Chrome; Windows 11	
Preconditions: The user has logged into the system.	
Test Case Steps: Basic flow	

Step No	Procedure	Expected Response	Pass/Fail
1	Choose a Space and navigate to this Space page, then click another user's username link in this Space.	Access the user's profile page and display the following and followers of this user.	pass
2	Click the "Follow" button of a user	A "FOLLOWS" relationship is created between the two users.	pass
3	Click the "Unfollow" button of the user who previously followed.	The "FOLLOWS" relationship from the current user to the following user is deleted.	pass
4	Click the "Block" button of a user.	A "BLOCKS" relationship is created from the current user to another user.	pass
5	Click the "Unblock" button for the previously blocked user.	The "BLOCKS" relationship from the current user to the blocked user is deleted.	pass
Comments: This function includes the user profile webpage.			

Table 8: Follow, unfollow, block and unblock other users function testing

6.2 Recommendation results

There isn't sufficient actual data for me to test, so I mainly show the recommendation results for some virtual users here and compare these results with the users' historical Space interaction data to determine whether the recommendation is effective. The virtual data here is created manually when I test the application without using code.

6.2.1 Test User 1

The user “NIETTANFT” has participated in the Spaces below:

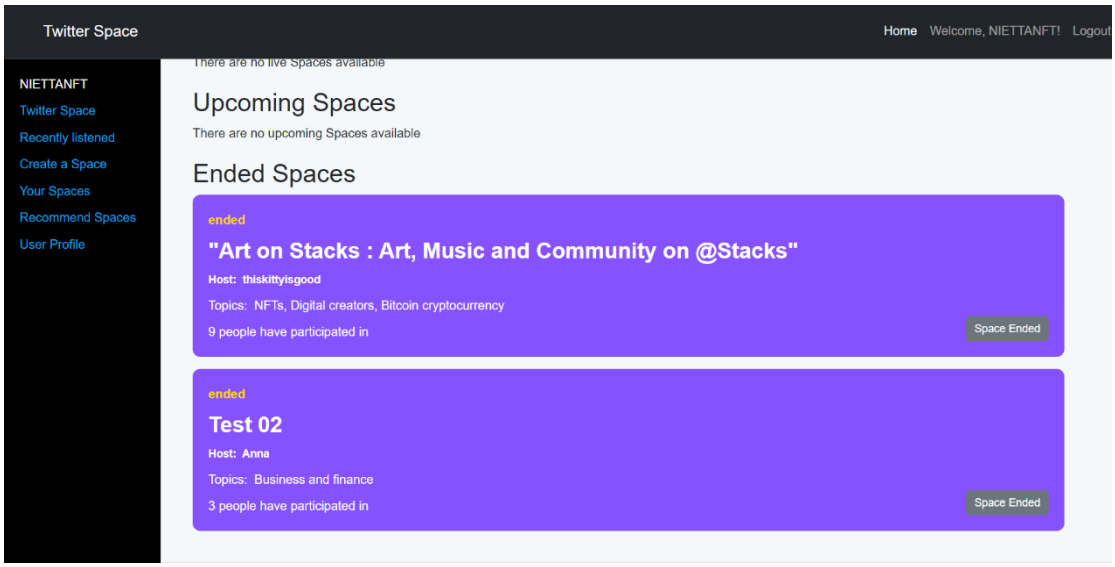

Figure 43: Spaces that “NIETTANFT” participated in

And the recommended Spaces for user “NIETTANFT” are shown as follows:

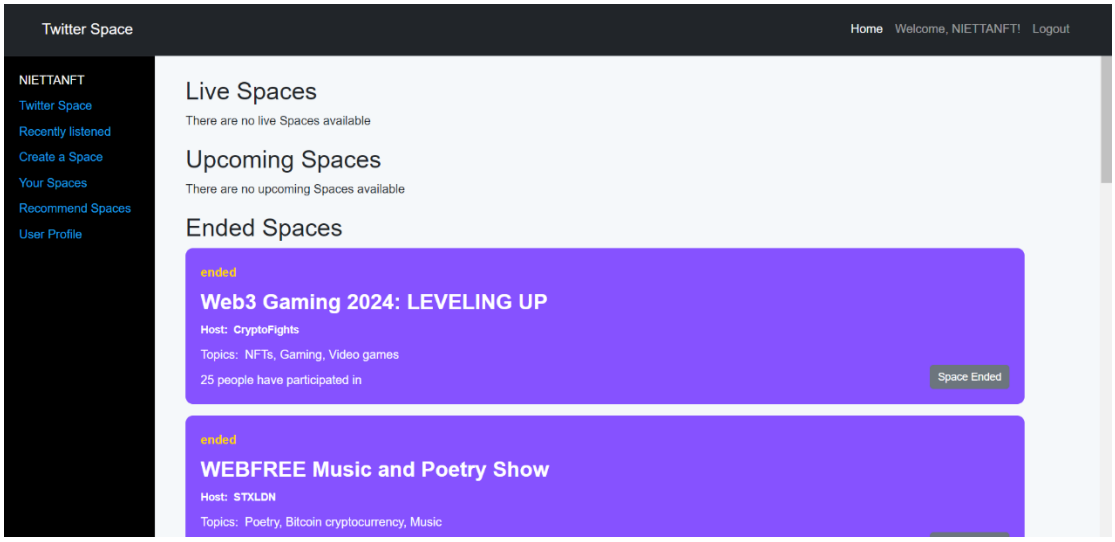

Figure 44: Recommended Spaces for “NIETTANFT”, part 1

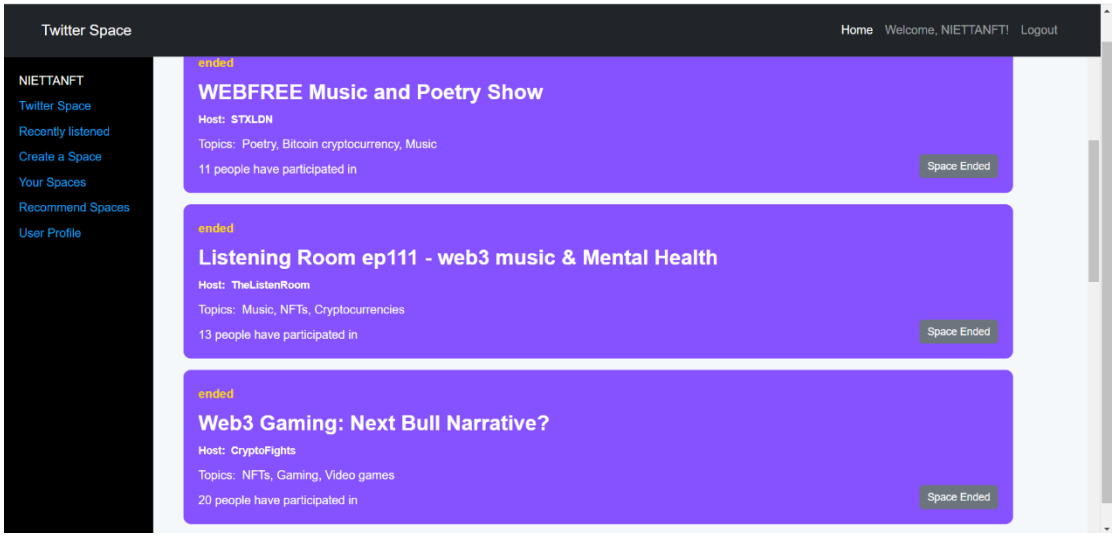

Figure 45: Recommended Spaces for “NIETTANFT”, part 2

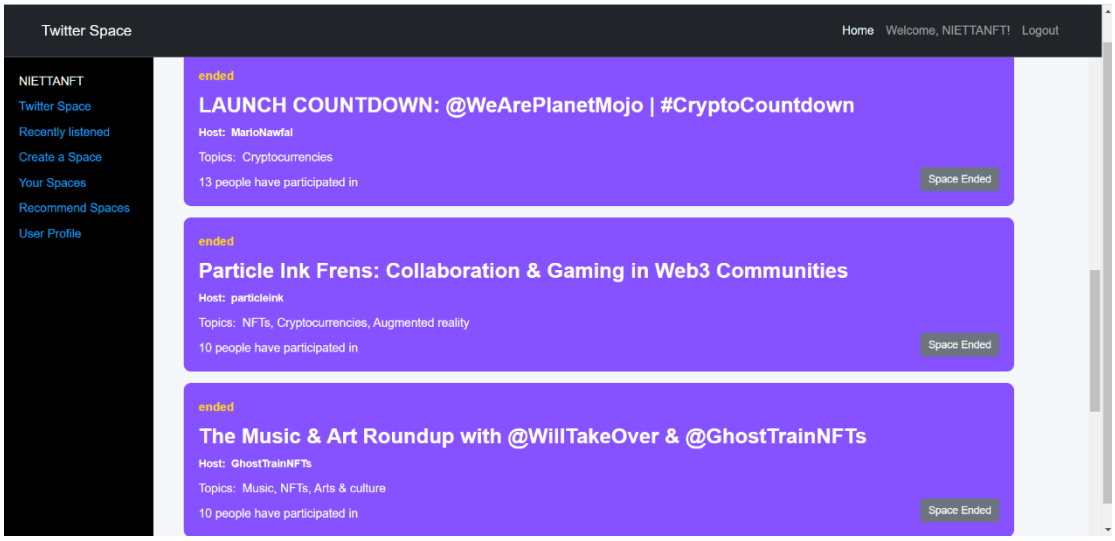

Figure 46: Recommended Spaces for “NIETTANFT”, part 3

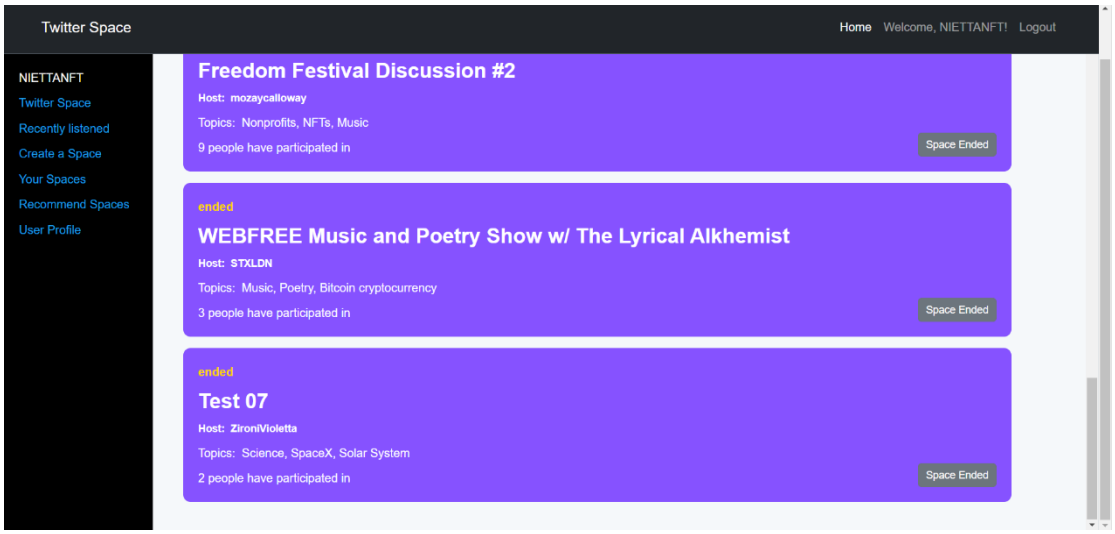

Figure 47: Recommended Spaces for “NIETTANFT”, part 4

6.2.2 Test User 2

The user “dhglover” participated Spaces are shown in the figures below:

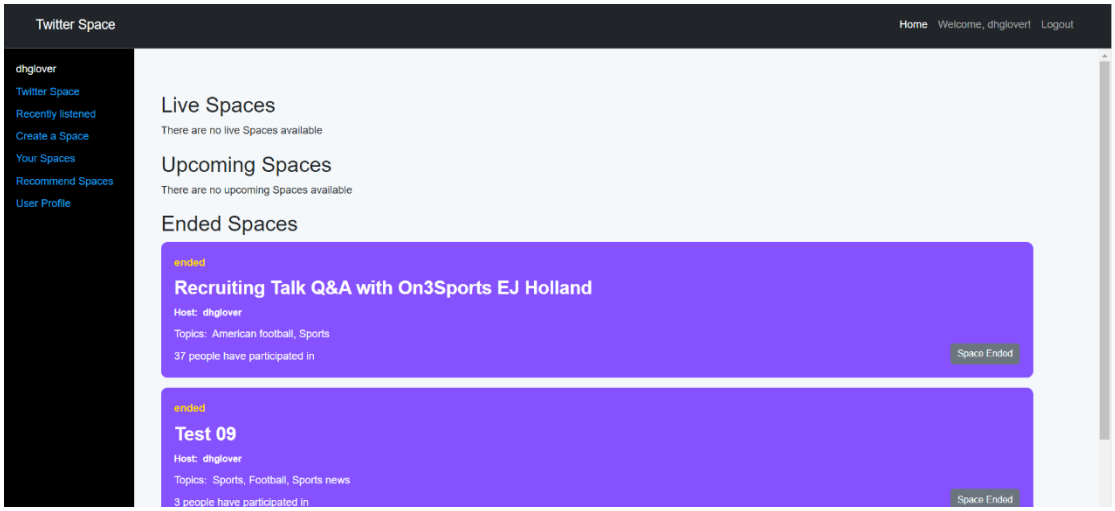

Figure 48: “dhglover” participated Spaces, part 1

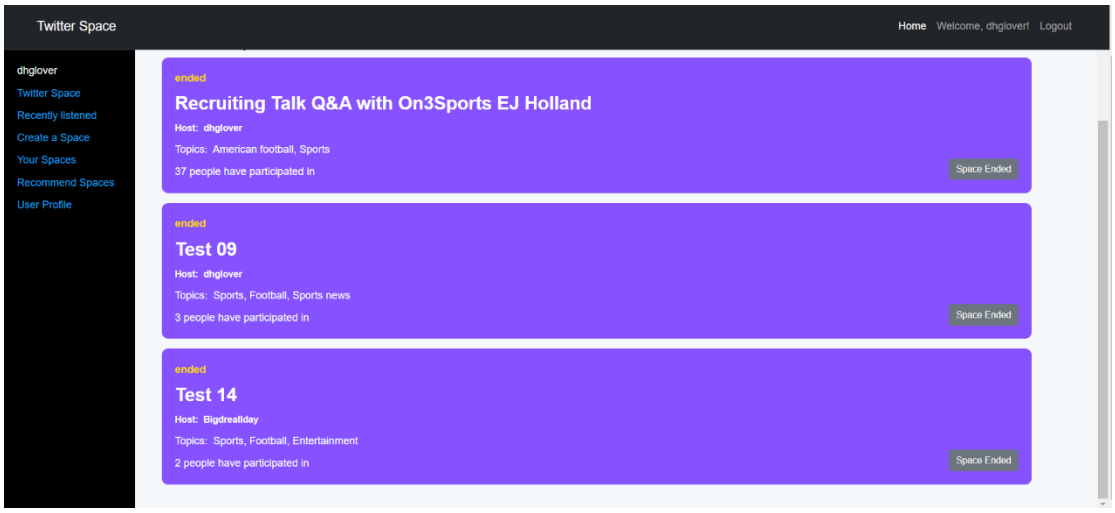

Figure 49: "dhglover" participated Spaces, part 2

The recommended Spaces for the user "dhglover":

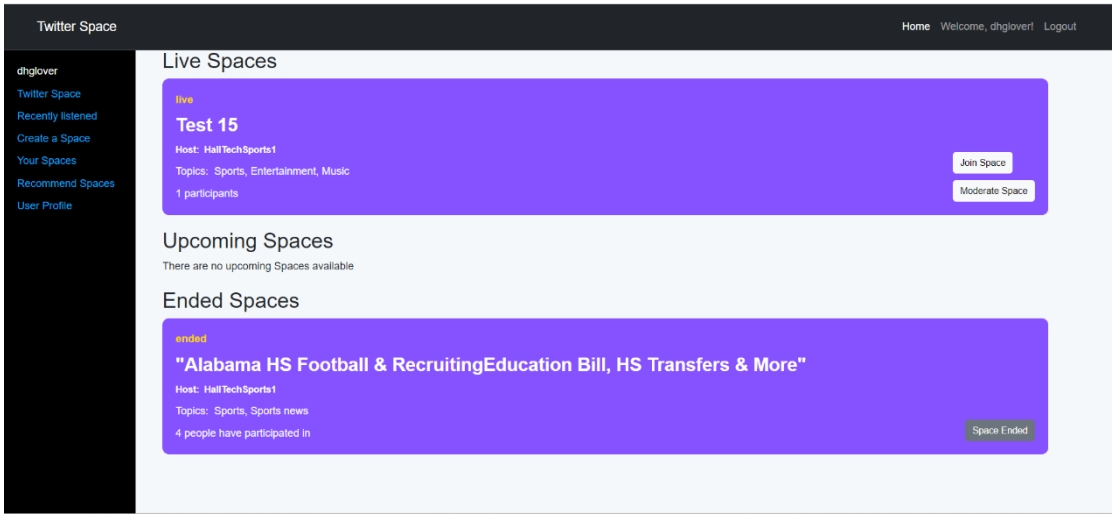

Figure 50: Recommended Spaces for "dhglover"

6.2.3 Test User 3

The user "DabbyCrypto" participated Spaces are shown in the figures below:

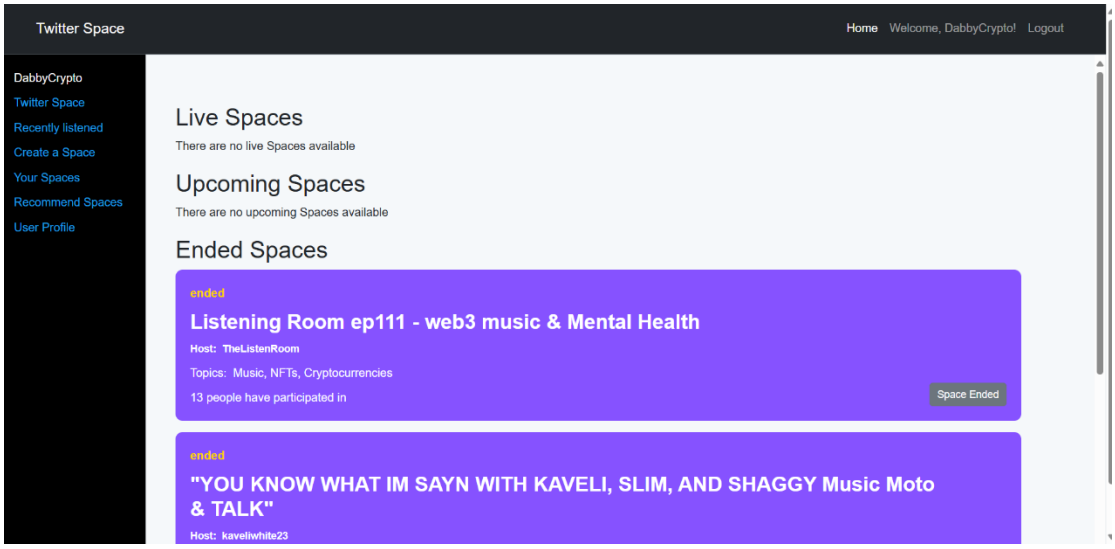

Figure 51: "DabbyCrypto" participated Spaces, part 1

Figure 52: "DabbyCrypto" participated Spaces, part 2

The user "DabbyCrypto" gets recommendations as following figures:

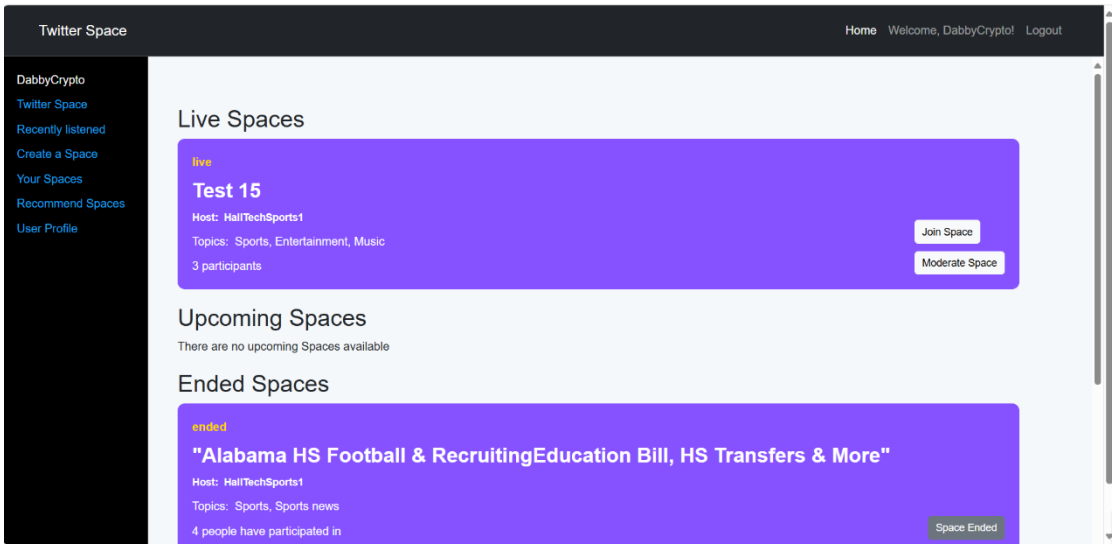

Figure 53: Recommended Spaces for "DabbyCrypto", part 1

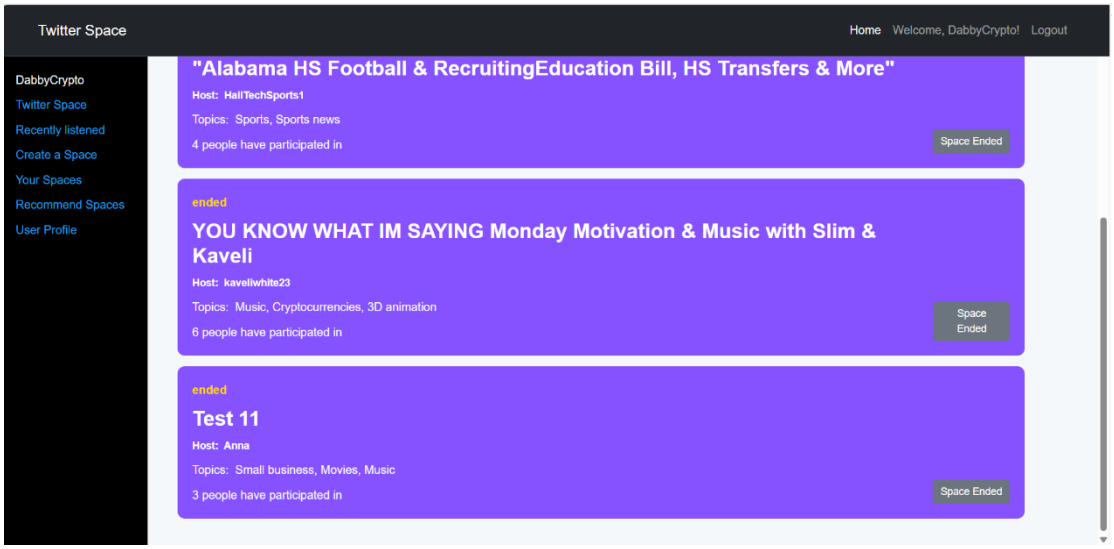

Figure 54: Recommended Spaces for "DabbyCrypto", part 2

Based on their participation, we can observe that some of the topics of the recommended Spaces align with those of the Spaces they have participated in. Furthermore, users can receive recommendations for items they have not previously expressed interest in but which similar users have shown interest in. This can be beneficial in identifying users' potential interests. Consequently, the recommendation engine is effective to some extent.

7. Discussion and Conclusion

The recommendation method can find users' potential interests by calculating the user similarity. This can be reflected in the recommendation results: some Spaces have topics that users did not show interest in, but similar users participated in these Spaces, so these Spaces can also be recommended to users. This phenomenon aligns with the long-tail effect; users can not only get recommendations that are popular in the system but also get unpopular recommendations through this system.

In addition, the recommendation method I used in the system, considering both the user's duration and the Space's duration, can make the recommended results more accurate to some extent, but it can be improved by improving the way topic weights are calculated. Fractions can lose some important information when determining users' degree of involvement, for example, considering a situation where one user's average participation time is 1 hour, but they spent 2 hours in a Space with a total duration of 4 hours; another user, who typically participates for 2 hours, also spent 2 hours in the same Space. Obviously, the first user may be more interested in this Space, but in my method, the weights could be the same.

This recommendation method also has some shortcomings. One of the shortcomings of this method is that it can only recommend Spaces that similar users have participated in, so the recommended results may include a few live Spaces. Additionally, another disadvantage is that when a Space has multiple topics, the user's interest value in these topics may be "diluted", especially when there is a lack of user interaction data. For instance, considering the situation shown in Figure 55, we can get a user-topic weight table shown in the following:

User	Topic 1	Topic 2	Topic 3	Cosine similarity with Anna
Anna		0.5	0.5	1
Luna	0.5	0.5		0.5
Jane			0.5	0.71

Table 9: The user-topic weight matrix

From Table 9, we can see that Anna is more similar to Jane, but from Figure 55, we can see that Anna may be more interested in Topic 2, so she may be more similar to Luna. As Space 3 has two topics, the topic weight has been “diluted". Hence, the recommendation results will be inaccurate when there are few user participation records.

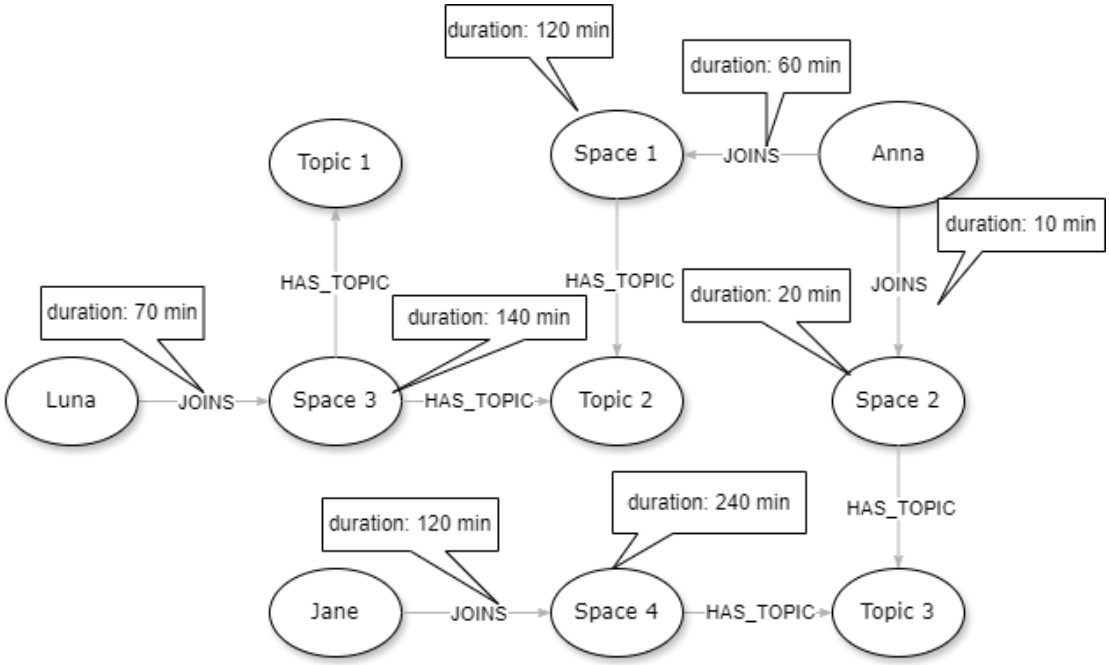

Figure 55: A data example

On the other hand, since the user can define topics, the topic vocabulary is uncontrolled; it can contain different words with the same meaning and polysemy resulting in ambiguity(Shoja and Tabrizi 2019).

In conclusion, this system implemented a basic recommendation engine, which is effective to some extent. In the future, this algorithm can be improved with more in-depth research.

8. Future works

As mentioned in the discussion section, my recommendation engine needs to be improved. In future works, I can try to deal with the problems mentioned in the

discussion section, such as using more appropriate methods to calculate the topic weight and deal with the missing information caused by fractions. In addition, audio content is an essential source for extracting Space features. Due to time constraints, I did not extract audio features in this project. In future works, I can discuss how to transcribe the audio content and then extract keywords from the transcribed content to make more accurate recommendations to users. Furthermore, I implemented the follow and block functions in this project but did not utilise this function in my recommendation algorithm. The user's social relationship is also important to the user profile. How to combine these factors with the recommendations implemented in this project is also a direction that can be expanded.

9. Reflection of Learning

In this project, I learned and understood how to use Neo4j, had a preliminary understanding of how to handle simultaneous two-way communication, and gained a deeper understanding of the HTTP and WebSocket protocols. Additionally, I became more familiar with the web development process by using Flask and knew how to arrange the file structure of a project.

I understood the basic recommendation methods in recommendation systems. I also learned several methods for calculating similarity and some mathematical knowledge. In the process of discovering problems and trying to solve these problems, I found some literature and tried to propose possible solutions from the literature.

I faced many challenges when I tried to find solutions for the recommendation engine. I was too eager to find a perfect solution, but while reading the literature, I found some content difficult to understand. I didn't understand the requirements of this project at the beginning, and I reviewed a lot of literature that was challenging for me at the time, but I didn't use them in the end. Although I encountered these challenges and some of my efforts seemed useless, I understood more about the recommendation system and knowledge graph through this process.

During the system development process, I also faced a series of challenges. For example, in the beginning, I spent much time trying to find a better way to develop the Twitter model. However, I eventually abandoned the Twitter model and focused on developing the Twitter Space model. To develop more quickly, I ignored requirements gathering at the beginning. In the middle of the development, I found that the functions I developed did not meet the requirements, so I revised them, which took more time. Therefore, when developing the system in the future, I need to collect detailed requirements at the beginning and then start developing after requirements gathering.

In short, during this project, I learned a lot that will be useful for future work, and I also experienced many setbacks, from which I gained valuable experience and lessons. I am satisfied with the knowledge and experience I gained from this project, and I will continue to learn and improve.

Reference

Bobadilla, J., Ortega, F., Hernando, A. and Gutiérrez, A. 2013. Recommender systems survey. *Knowledge-Based Systems* 46, pp. 109–132. Available at: https://linkinghub.elsevier.com/retrieve/pii/S0950705113001044 [Accessed: 20 August 2024].

Carman, A. 2021. *Facebook's Clubhouse competitor starts rolling out in the US today*. Available at: https://www.theverge.com/2021/6/21/22540508/facebook-live-audio-rooms-clubhouse-rival-spaces-greenhouse [Accessed: 30 September 2024].

Chicaiza, J. and Valdiviezo-Diaz, P. 2021. A Comprehensive Survey of Knowledge Graph-Based Recommender Systems: Technologies, Development, and Contributions. *Information* 12(6), p. 232. Available at: https://www.mdpi.com/2078-2489/12/6/232 [Accessed: 10 August 2024].

Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H. and He, Q. 2022. A Survey on Knowledge Graph-Based Recommender Systems. *IEEE Transactions on Knowledge and Data Engineering* 34(8), pp. 3549–3568. Available at: https://ieeexplore.ieee.org/document/9216015/ [Accessed: 8 August 2024].

jeremiah_owyang. 2021. *The Future of Social Audio: Startups, Roadmap, Business Models, and a Forecast*. Available at: https://web-strategist.com/blog/2021/01/30/the-future-of-social-audio-startups-roadmap-business-models-and-a-forecast/ [Accessed: 17 August 2024].

Ji, S., Pan, S., Cambria, E., Marttinen, P. and Yu, P.S. 2022. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems* 33(2), pp. 494–514. Available at: https://ieeexplore.ieee.org/document/9416312/ [Accessed: 10 August 2024].

Lathia, N., Hailes, S. and Capra, L. 2008. The effect of correlation coefficients on communities of recommenders. In: *Proceedings of the 2008 ACM symposium on Applied computing*. Fortaleza, Ceara Brazil: ACM, pp. 2000–2005. Available at: https://dl.acm.org/doi/10.1145/1363686.1364172 [Accessed: 31 August 2024].

Leskovec, J., Rajaraman, A. and Ullman, J.D. 2019. *Mining of Massive Datasets*. 3rd ed. Cambridge: Cambridge University Press

Levy, A., Shalom, B.R. and Chalamish, M. 2024. *A Guide to Similarity Measures.* Available at: http://arxiv.org/abs/2408.07706 [Accessed: 27 August 2024].

NumFOCUS. 2024. *pandas - Python Data Analysis Library*. Available at: https://pandas.pydata.org/ [Accessed: 24 August 2024].

Pokorný, J. 2015. Graph Databases: Their Power and Limitations. In: Saeed, K. and Homenda, W. eds. *Computer Information Systems and Industrial Management*. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 58–69. Available at: http://link.springer.com/10.1007/978-3-319-24369-6_5 [Accessed: 16 August 2024].

Ricci, F., Rokach, L. and Shapira, B. eds. 2022. *Recommender Systems Handbook*. New York, NY: Springer US. Available at: https://link.springer.com/10.1007/978-1-0716-2197-4 [Accessed: 19 August 2024].

Shoja, B.M. and Tabrizi, N. 2019. Tags-Aware Recommender Systems: A Systematic Review. In: *2019 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering (BCD)*. Honolulu, HI, USA: IEEE, pp. 11–18. Available at: https://ieeexplore.ieee.org/document/8884850/ [Accessed: 5 September 2024].

Webber, J. 2021. Powering Real-Time Recommendations with Graph Database Technology. Available at: https://go.neo4j.com/rs/710-RRC-335/images/Powering-Recommendations-Neo4j-white-paper-EN-A4.pdf [Accessed: 5 September 2024].