VoiceJive: A Voice-Powered Social Media App for Idea Sharing - Implementation Using Firebase and Flask

Issam Seddik[1] and Jamal Boussouf[1]

[1]Multidisciplinary Faculty of Nador - Master in Data Science and Intelligent Systems (SDSI)
{*sedissam, jamalbssouf*}*@gmail.com*

July 18, 2023

Abstract

This work introduces a Flask-based social media application using Firebase for idea sharing and user interaction. Users can share thoughts through voice recordings with captions, fostering meaningful online interactions. The application includes user management, post creation, friend connections, and a user-friendly interface. Flask and Firebase enable seamless integration, real-time data synchronization, secure authentication, and efficient storage. This work contributes to enhancing social media platforms by providing an interactive medium for idea sharing and improving user experiences.

Keywords: Social media, Flask, Firebase, voice recordings, captions, idea sharing, user interaction.

Figure 1: The principal tools used

Introduction

In the era of social media dominance, the need for innovative platforms to foster idea sharing and engagement is becoming increasingly important. Traditional text-based posts are often limited in their ability to convey the depth and nuance of ideas. To address this limitation, we present VoiceJive, a novel social media app that enables users to share their ideas in the form of voices, accompanied by captions. Leveraging the power of voice communication, VoiceJive aims to enhance user experiences and facilitate more meaningful interactions within the online community. The implementation of VoiceJive relies on two key technologies: Firebase (NoSQL) and Flask. Firebase provides a robust backend infrastructure, offering real-time database capabilities, secure user authentication, and scalable cloud storage. Flask, a lightweight web framework, allows for seamless interface development and efficient routing between various app components.

By integrating voice recording functionality within VoiceJive, users can effortlessly record their thoughts and ideas, which are then stored and made accessible to other users in a highly engaging format. The addition of captions further enhances the accessibility and discoverability of the shared content, enabling users to search, filter, and explore ideas effectively. Firebase's real-time database features ensure that VoiceJive stays up-to-date with the latest user-generated content. This allows for immediate interaction and feedback from the community, fostering a dynamic environment for idea sharing and collaboration. Additionally, Firebase's secure user authentication safeguards user data, ensuring a safe and trusted platform for users to express themselves. The Flask framework enables the development of a user-friendly interface for VoiceJive, seamlessly integrating voice recording and captioning features. It provides a flexible and extensible platform for implementing various functionalities, such as content discovery, user profiles, and social interactions, all designed to enhance the user experience. Our paper presents the implementation of VoiceJive, a voice-powered social media app that leverages Firebase and Flask to facilitate the sharing of ideas through voice recordings and captions. By combining the power of voice communication with the flexibility of text-based content, VoiceJive aims to provide users with an engaging and immersive platform for idea sharing, fostering creativity, and encouraging meaningful interactions within the online community.

1 User Interface

The user interface of the Flask-based social media application plays a crucial role in providing a seamless and engaging user experience. It encompasses various components and features that allow users to interact with the application and perform different tasks. The user interface is implemented using HTML templates, CSS styles, and Flask's template rendering capabilities, ensuring dynamic generation of HTML pages based on user interactions.

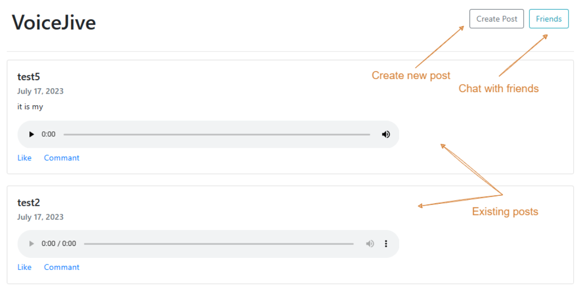


Figure 2: User interface screenshot

1.1 Login and Signup Pages

The application's user interface begins with a login page, where users can enter their credentials to access their accounts. In case users do not have an account, a signup page is available for new user registrations. These pages typically include input fields for email addresses, passwords, and submit buttons for form submission. The UI design ensures a clean and intuitive layout, making it easy for users to enter their information and proceed with the authentication process.

1.2 Home Page

Upon successful login, users are directed to the home page, which serves as the main dashboard of the application. The home page provides an overview of the user's activities, including their posts, friends' updates, and relevant notifications. It typically displays a feed of posts from the user's friends, allowing users to browse through and engage with the content. The UI design focuses on presenting the content in an organized and visually appealing manner, enhancing the user's browsing experience.

1.3 Post Creation

To encourage users to share their ideas, the application includes a post creation feature accessible from the home page. This feature enables users to create new posts by uploading audio files and adding captions. The user interface for post creation includes an audio file upload field, a text input field for captions, and a submit button to publish the post. The design emphasizes simplicity and ease of use, allowing users to effortlessly create and share their voice recordings with the community[7][9].

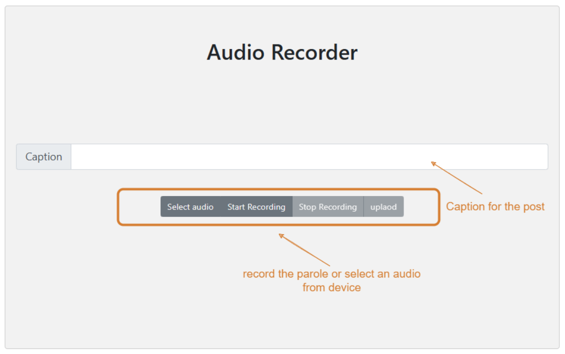


Figure 3: Create post screenshot

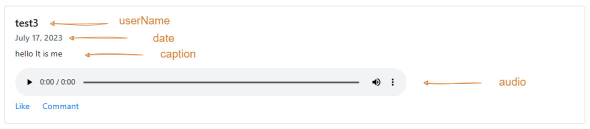
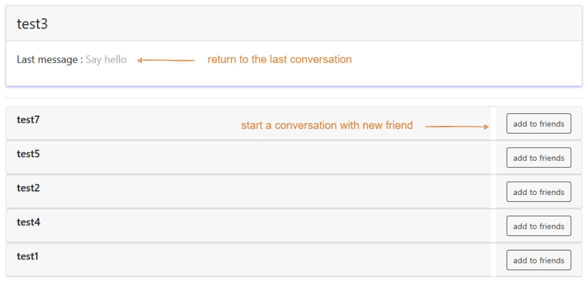


Figure 4: Post screenshot

1.4 Friends and Connections

The application's user interface provides features for managing friends and connections. Users can view their list of friends The UI design ensures clear visibility of friend suggestions, pending requests, and accepted friend connections. Users can navigate through their friends', view their posts, and interact with their content (likes and comments). The interface facilitates seamless social interactions, fostering connections and collaborations within the user community[10].

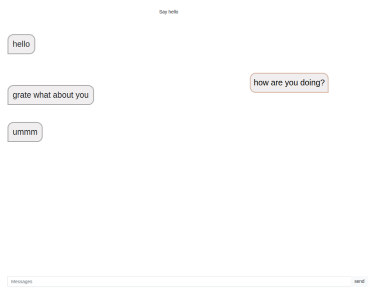

test3

Last message : Say hello ← return to the last conversation

test7	start a conversation with new friend →	add to friends
test5		add to friends
test2		add to friends
test4		add to friends
test1		add to friends

Figure 5: List of friends

the following figures show the graphical interface used for communication with users:


Say hello

hello

how are you doing?

grate what about you

ummm

Figure 6: List of friends

2 Architecture

The project is implemented using Python and utilizes the following libraries and frameworks:

- firebase: A Python library for Firebase, used for authentication and accessing the Firestore database.
- post: A module for handling posts, including likes and comments.
- user: A module for managing all the functionality of the user on the app.
- audio: A module for managing audio files and uploading them to Firebase Storage. datetime: A Python library for manipulating dates and times.

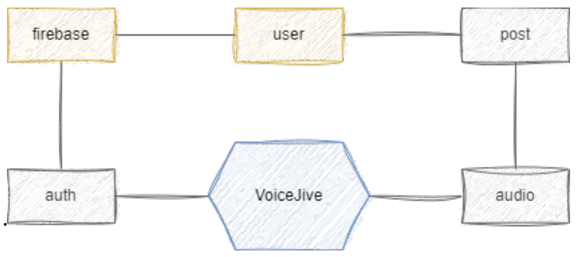


Figure 7: Overall system architecture

The project is structured as a collection of functions and a User class. The functions and methods interact with the Firebase services to perform various operations related to user management, post creation, friend management, and messaging.

3 Functionality

3.1 User Management

The user management functionality of the VoiceJive project provides several useful methods for retrieving user-related data. The get_ref(user_id) function retrieves the Firestore document reference for a given user ID, allowing easy access to the user's data. The get_posts(uid) function retrieves the posts associated with a user by fetching the posts from the Firestore database. Similarly, the get_UserName(uid) function retrieves the username of a user from the Firestore database.

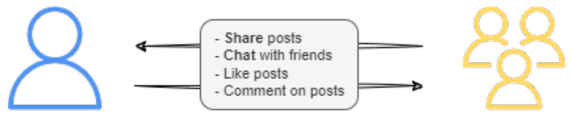


Figure 8: All user functionality

The getAll_posts() function is responsible for retrieving all posts from all users. It utilizes the Firebase Admin SDK's auth.list_users() method to obtain a list of all registered users. It then iterates over each user, retrieves their posts and username using the previously defined functions, and aggregates them into a list of tuples representing the username and post content[8] [11].

The get_friend(uid) function retrieves the friends of a user by accessing the Firestore database and retrieving the friend list associated with the user. This function returns the friend list as a dictionary, where the keys are the friend IDs and the values contain additional information about the friend.

The last_msg(uid, key) function retrieves the last message between the user and a specific friend. It accesses the Firestore database, retrieves the friend's messenger list, and returns the last message in the list.

The get_all_friends(uid) function retrieves information about all friends of a user.
It uses the get_friend(uid) and last_msg(uid, key) functions to obtain the friend list and last message for each friend. The function returns a list of lists, where each sublist contains the friend ID, username, and the last message exchanged between the user and that friend.

The not_friends(uid) function retrieves a list of users who are not friends with the given user. It first obtains a list of all registered users using the Firebase Admin SDK's auth.list_users() method. It then compares this list with the friend list of the user, obtained using the get_friend(uid) function, to identify users who are not friends. The function returns a list of tuples containing the user ID and username for each non-friend user.

3.2 Post Creation

The post creation functionality allows users to create new posts, add likes to existing posts, and comment on posts. The functionality is encapsulated within the User class.

The create_post(content) method of the User class is responsible for creating a new post for the user. It takes the content of the post as input, including an audio file. The method first retrieves the user's existing posts from the Firestore database and increments the audio file's index to ensure a unique filename. It then uses the FirebaseStorageManager from the audio module to upload the audio file to Firebase Storage. The method updates the post content with the storage URL of the audio file and adds the new post to the user's list of posts in the Firestore database[1][2].

The like_post() method allows the user to like a specific post. This method invokes the add_like(self) method of the post module, passing the user as an argument. The add_like(self) method handles the addition of the user's like to the post, updating the likes count and the list of users who liked the post.

The comment_on_post(post, comment) method enables the user to comment on a specific post. Similar to the like_post(post) method, it invokes the add_comment(self, comment) method of the post module, passing the user and the comment as arguments. The add_comment(self, comment) method adds the user's comment to the post, maintaining a list of comments associated with the post[3] [4].

3.3 Data Retrieval

The data retrieval functionality includes the usertodict() method, which retrieves the user data from Firestore and returns it as a dictionary. This method accesses the Firestore database, retrieves the document for the given user ID, and converts the document snapshot into a dictionary representation of the user's data[5].

4 Data Structure in Firestore

The data in Firestore for the VoiceJive project is organized using a specific structure.

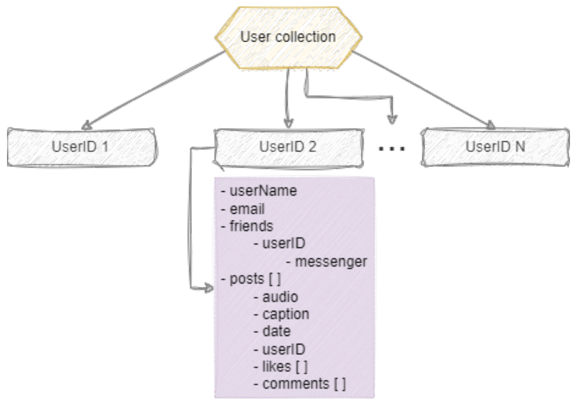


Figure 9: User collection and data structure

In this updated structure, the messenger field is nested within each friend's entry in the friends subcollection. Each friend is represented as a document with a unique friend ID, such as "l8Kf1eDi8...". The messenger field within each friend document stores the messaging history between the user and that specific friend[8][12].

The revised structure reflects the relationship between friends and their associated messaging history. It allows for easy access to messaging data within the context of each friend, facilitating efficient retrieval and management of messages between users in the VoiceJive project. Below is an example of the structure of the data stored in Firestore[6]:

```

Document: JRGSYWXX8SSK3DzXWMvjwy0KvpE2
  - email: "test2@gmail.com" (string)
  - friends:
     - l8Kf1eDi8yeozqO0VQJu1xj22gP2:
        - messenger:
           - 0: "sys/say hello"
     - q5M9ze029jU5U1v5ftlcbp1jb4i1:
        - messenger:
           - 0: "sys/say hello"
  - posts:
     - 0:
        - audio: "link to audio"
        - caption: ""
        - date: "July 17, 2023"
        - uid: "JRGSYWXX8SSK3.."
        - likes: ['uid1', 'uid2']
        - comment: [('uid1', 'hello')]
  - userName: "test2"
```

5 Conclusion

The VoiceJive project provides a robust foundation for a social media platform, incorporating user management, post creation, friend management, and messaging functionalities. Users can retrieve their posts, view their friends and messages, create posts with audio files, interact with posts through likes and comments, add friends, and exchange messages. The project utilizes Firebase for authentication, data storage, and real-time updates, ensuring a secure and scalable platform.

With further development, the VoiceJive project can be expanded to include additional features such as notifications, user profiles, privacy settings, and advanced search functionalities. The flexible architecture and modular design of the project allow for easy integration of new features and enhancements to meet the evolving needs of the users.

References

[1] Firebase admin sdk documentation. `https://firebase.google.com/docs/admin/setup`. Accessed on [Access Date].

[2] Firebase authentication documentation. `https://firebase.google.com/docs/auth`. Accessed on [Access Date].

[3] Firebase documentation. `https://firebase.google.com/docs`. Accessed on [Access Date].

[4] Firebase real-time database documentation. `https://firebase.google.com/docs/database`. Accessed on [Access Date].

[5] Firebase storage documentation. `https://firebase.google.com/docs/storage`. Accessed on [Access Date].

[6] Firestore documentation. `https://firebase.google.com/docs/firestore`. Accessed on [Access Date].

[7] Flask-bcrypt documentation. `https://flask-bcrypt.readthedocs.io/`. Accessed on [Access Date].

[8] Flask documentation. `https://flask.palletsprojects.com/`. Accessed on [Access Date].

[9] Flask-sqlalchemy documentation. `https://flask-sqlalchemy.palletsprojects.com/`. Accessed on [Access Date].

[10] Flask-wtf documentation. `https://flask-wtf.readthedocs.io/`. Accessed on [Access Date].

[11] Jinja documentation. `https://jinja.palletsprojects.com/`. Accessed on [Access Date].

[12] Python documentation. `https://docs.python.org/`. Accessed on [Access Date].