Hello, Friends! Draft Project

Vision Statement:

Member names / emails:

1. Tiange Wang / tiange@ucsb.edu
2. Changwei Xu / c_xu@ucsb.edu
3. Jianyu Yu / jianyuyu@ucsb.edu
4. Yifan Yu / yifanyu@ucsb.edu
5. Derek Bang / dkbang@ucsb.edu

This project is a 1-on-1 video chat app that provides translation capabilities. The app will include a main video screen that projects the webcam video feed and several APIs working in the background that provides speech-to-text and text translation functions. The users on both sides first choose the intended language that they would like to use and the language that they wish to translate to, then they can start the video and audio feed where they will be speaking in the language they chose at the beginning and the APIs will process this speech and translate it into the intended language that the user also specified at the beginning. It will be built in Java and the intended platform is PC.

This project has 2 intended user groups. The first group is people hoping to make friends around the world in order to learn their respective cultures and languages and the second group is for business people where communicating in different languages among different teams in a meeting can present a problem. Today, there are solutions provided for both
groups, but they are not suitable for every scenario. For the first group of people, meeting people that are different from them is interesting but it could be difficult for them to communicate. There is online translation software for translating, however, they cannot be conducted in real time which hinders their ability to communicate effectively. For the second group, it is essential for different teams around the world to speak in a language that is understandable so their projects can be carried out without trouble. Even though translators can be provided, they are not always available and it could be difficult to communicate via another person.

Video calls would be possible among people speaking different languages. Nowadays, video calling software such as Facetime, Zoom is able to satisfy the daily video chat needs. This software could satisfy people’s daily requirement of video calling, shorten the distance among people. Employees who work hundreds of miles away from their families could meet easily, despite the limitation of distance. Companies also use video meeting software to boost the efficiency of discussing. With the help of real-time video streaming, remote communication has become swiftly, no longer merely based on language.

However, there are still a lot of video chats strongly dependent on language. For example, How cannot people with different languages communicate with each other fluently? How to convey instructions to foreign people clearly and accurately via video chat? These are the major problems that Video Calling Software need to address and solve. Therefore, the function of our software is extremely useful and can fill the huge gap in the current Video Calling Software market.
**Timeline:** Research → Basic Structure → Basic Front-End Design→ Basic Front-End Implementation(Using Java swing GUI) → More Research(Mostly on Back-End) → integrate 1-to-1 text chat → integrate video chat (openvidu api) → integrate voice to text (google cloud speech api) → integrate translation(google translate api) → Initial Testing & Polish → additional features such as better front-end, text chat, scribbles to text, watching youtube videos together w/translation, group chat) → Final Testing & Polish → Final Product

---

**Links:**

1. Github: [https://github.com/TgWang1023/Hello-Friend](https://github.com/TgWang1023/Hello-Friend)
2. Travis-ci: [https://travis-ci.org/TgWang1023/Hello-Friend](https://travis-ci.org/TgWang1023/Hello-Friend)
3. Slack: [https://hellofriendtalk.slack.com](https://hellofriendtalk.slack.com)

---

**System Architecture Overviews**

Class Diagram:

[https://drive.google.com/open?id=1UOdFUdQngMRnX9kvMpC88Rgk46k0g8v](https://drive.google.com/open?id=1UOdFUdQngMRnX9kvMpC88Rgk46k0g8v)
Requirements:

Case 1: Create Chat Room

User story 1: As a user, I can create a chat room so that another user can join my chat and talk to me.

Actors: User, Server, Manager

Precondition: The user has connected to the server, fills and submits the room creation request from.

Flow of events:

Basic Path:
1. Server receives the room-creation request from the user
2. Controller handles the form and calls the create function
3. Controller calls Manager class to create a room with given name
4. If the creation is successful, Manager will return success code to Controller and set up the chat channel
5. The user will receive a success message from Server

Alternative Paths:
1. If the user submits an incomplete form, then the server will return an error message “incomplete form”. (exception case)
2. If the user submits a room name that has been occupied, then the server will return an error message “occupied room name”. (exception case)

Postcondition: The user can now use the channel for chat or he/she has to submit another form again (if creation fails)

Case 2: Join Chat Room with Room Name

User story 2: As a user, I can join a chat room so that I can start a private chat with the room owner.

Actors: User, Server, Manager

Precondition: The user has connected to the server, fills and submits the room join request form.
Flow of events:

*Base Path:*

1. Server receives the room-joining request from the user
2. Controller handles the form and calls the join function
3. Controller calls Manager class to join a room with given name
4. If the joining is successful, Manager will return success code to Controller and register new user in the chat channel
5. The user will receive a success message from Server

*Alternative Paths:*

1. If the user submits an incomplete form, then the server will return an error message “incomplete form”. *(exception case)*
2. If the user submits a room name does not exist, then the server will return an error message “no such room”. *(exception case)*
3. If the user attempts to join a room that is full (already has two chatters), then the server will return an error message “room is full”. *(exception case)*

*Postcondition:* The user can now chat in the channel or he/she has to submit another form again (if joining fails)

Case 3: Language Translation

**User story 3:** As a user, I can send a message in my preferred language so that the server can recognize and translate my message.

**Actors:** User 1, User 2, Server

**Precondition:** User has entered his/her language of choice into the session page and the chat session has started

**Flow of events:**

*Basic Path:*

1. User 1 enters message into the text box and submits to the server
2. Server receives the message and sends API request to Google Translate to translate the message into user 2’s preferred language
3. Server receives the translated message from the API and sends the translated message to user 2
4. User 2 receives the translated message

*Alternative Paths:*
1. User 1 enters empty/invalid message into the text box and attempts to submit to the server
2. Server receives the message and finds the error. Sends error message back to user 1
3. User 1 receives error message

*Postcondition:*
User 1’s message is translated into the preferred language of user 2 and displayed onto user 2’s interface or message is not sent and error message is displayed. (end-early path)

---

**Case 4: Disconnect Chat Room**

**User story 4:** As a user, I can disconnect from the server so that I will be removed from the chat room.

**Actors:** User 1, User 2 *(not necessary)*, Server, Manager

**Precondition:** User is current in a chat session

**Flow of events:**

*Basic Path:*
1. User clicks on “disconnect” to attempt to disconnect from the chat session and the client stops to receive messages from Server.
2. Server receives the disconnect request and removes the user from the channel.
3. If there is no user in the room, delete the room; otherwise send a system message to notify the other user.

**Postcondition:** The quitting user successfully disconnects from the Server, and the Server will not send messages to the user’s address.
Case 5: Video/Audio Chat

User story 5: As a user, I can send video and audio feed to the chat session so that another user in the current chat session can receive my feeds.

Actors: User 1, User 2, Server

Precondition: User has entered his/her language of choice into the session page and the chat session has started

Flow of events:

Basic Path:
1. User 1 turns on his/her webcam/microphone to allow video and audio transmission.
2. Video feed is passed directly from front-end into the OpenVidu API to establish RTC connection.
3. Once the connection between user 1 and user 2 has been established, feeds are transferred automatically
4. User 2 receives video/audio feed from user 1

Alternative Paths:
1. User 1 turns on his/her webcam/microphone to allow video and audio transmission.
2. Video feed is passed directly from front-end into the OpenVidu API and attempts to establish RTC connection but fails.
3. Error status is returned to the front-end and displayed on user 1’s interface.

Postcondition: User 2 receives real-time video/audio feed from user 1 or error message is displayed on user 1’s interface.(end-early path)

Appendix:

List of technologies employed: Java, HTML5, CSS3, Javascript(ES6), Apache Maven, Spring Framework, Google Translate API, Google speech-to-text API, OpenVidu API, Git, travis-ci