

Conversational Agent for Multilingual Groups

Fall 2022

Advisor: Dr. Susan Fussell

Coordinator: Jieun Kim

Morgan Cupp (mmc274@cornell.edu)

Berk Erdal (be236@cornell.edu)

Sidhee Hande (sph75@cornell.edu)

Anjali Kulkarni (avk57@cornell.edu)

Shreyas Shukla (ss3259@cornell.edu)

| | |
|-------------------------------|-----------|
| Introduction | 3 |
| System Interactions | 4 |
| User Interface Screens | 8 |
| Development | 17 |
| Database Structure | 17 |
| API Calls | 19 |
| Environment Setup | 30 |
| Frontend Setup | 30 |
| Backend Setup | 30 |
| REST Server | 30 |
| Event Publishing Server | 31 |
| Database Setup | 31 |
| IBM Cloud Setup | 31 |
| Deployment | 32 |

Introduction

The Conversational Agent for Multilingual Groups project aims to design and build a conversational agent to promote equal participation between native and non-native English speakers in multilingual groups. The participants must together complete the “Desert Survival” activity. The activity starts with the participants being told to imagine that they are survivors of a plane crash in the Sonoran desert and only have five items with them to survive. They must now decide on a ranking for the five items based on their relative importance for their survival in the desert. Each participant must first complete this activity on their own. Once all the participants have completed the activity, they are directed to a common screen where they must discuss and come to a consensus about what the final ranking should be. Once they reach a consensus, one of the participants will submit the final ranking for the team.

The participants' discussion is monitored by the researcher. Using the agent, the researcher can monitor how much each person is speaking and interject messages to motivate people to speak up. The presence of the researcher is unknown to the participants, who believe they are interacting with a fully autonomous system. This idea is known as a Wizard of Oz experiment. In conversations, non-native speakers often find it difficult to interject or find an appropriate place to join the conversation. This leads to one-sided conversations and the non-native speaker feeling left out. The researcher observes the conversation and views real time metrics of how the meeting is progressing in the form of the number of words each person has spoken, the time that they have been silent and the number of turns they took to speak. The researcher can send interjections to each participant separately without disturbing the ongoing flow of conversation.

The Conversational Agent uses a frontend developed using ReactJS and a backend developed in Python for all user (researcher and participant) interactions. Data is stored in a MongoDB database. Several external libraries were used to develop the agent such as NLTK, React-Beautiful-DND, React speech-to-text, Deepstream, etc.

The researcher has total control of the meeting and can act as a moderator. Overall, this project would prove quite successful in promoting equal participation among native and non-native English speakers.

System Interactions

Overall flow of interaction of researcher and participants with the system:

1. The researcher creates a meeting with a meeting ID.
2. The researcher shares the meeting ID with the participants.
3. The participants enter the meeting using the meeting ID.
4. The participants are presented with the Individual Decision Making Task Screen where they read the Desert Survival Prompt and then are asked to arrange five items from highest to lowest priority.
5. The participants can drag individual items from the bottom box, and drop it into individual slots on the top box. The participants can rearrange items within the top box as well.
6. Once participants arrange all five items, they are allowed to submit their individual responses.
7. Once a participant submits their individual task, they are redirected to a waiting room.
8. This is where the admin screen on the researcher's end comes into play. The researcher can see the names of participants who have submitted individual rankings and use this information to decide when to start the group discussion.
9. Once the researcher starts the group discussion, participants are redirected from the waiting page to the Group Decision Making Task page.
10. The meeting proceeds with participants having a conversation, trying to reach a consensus for the priority of items. During the meeting, the participants' voices will be recorded and used for near-real time statistics like time being silent, word count, etc.
11. During this time, the researcher can see the word counts of each participant, along with the number of turns they took to speak and also the time they were silent. The researcher can also see who has left the meeting at any time during the discussion.

12. For the meeting to end, the researcher has to end the meeting for all participants using the admin console. Individual participants can leave the meeting from their end as well.
13. After the meeting ends, every participant is redirected to the survey page and the researcher has the option to get a summary of the meeting and keywords used in the meeting.

Detailed flow of researcher's interactions with the system:

1. The researcher navigates to the create meeting [url](#) and is greeted with a page that is modeled in Figure 9.
2. The researcher enters a meeting ID and tries to create a meeting
 - If the meeting ID was used before, an alert will show up asking to enter a new, unused meeting ID.
 - If the meeting ID was not used before, the system will create a meeting with the specified meeting ID.
3. Once the meeting is created, the researcher will be redirected to the admin page as shown in Figure 10. The researcher is then able to inform the participants of the meeting ID.
4. Once all the participants join the meeting, even before starting the group discussion the researcher will see them in the “*send to*” drop down menu, as shown in Figure 10-12.
5. Once a participant submits the individual decision task, the researcher will see their name displayed at the top of the screen as shown in Figure 11.
6. The researcher can then start the group discussion. Once the researcher does so, a timer will be initiated and all the participants will be redirected to the group decision task making screen from the waiting room.
7. The researcher can send participants interjection messages through this page as necessary.
8. The researcher can also end the meeting at any time by selecting the end meeting button. This will end the meeting for all the participants, and all the participants will be redirected to the survey page as shown in Figure 8.

9. The researcher can view a complete summary of the meeting or breakdown of meeting keywords by clicking on the respective buttons. This is shown in Figure 14.

Detailed flow of participant's interaction with the system:

1. The researcher shares the meeting ID with the participant.
2. The participant navigates to the participant [url](#).
3. The participant reads the instructions and proceeds by pressing the 'Begin' button as shown in Figure 1.
4. The participant will then enter their *name*, *net ID*, and *the meeting ID* as specified by the researcher. They can proceed only after they accept all the other fields.
 - If the participant has entered a wrong meeting ID or the meeting has not been created yet, the participant will see an error message.
5. The participants are presented with the Individual Decision Making Task Screen where they read the Desert Survival Prompt, and then are asked to arrange five items according to what they will prioritize.
6. The participants can drag individual items from the bottom box, and drop it into individual slots on the top box. The participants can rearrange items within the top box as well.
7. Once participants arrange all five items, they are allowed to submit their individual responses.
8. Once a participant submits their individual task, they are redirected to a waiting room.
9. The participants are redirected to the Group Decision Task Screen when the researcher starts the meeting. Once they are redirected to the Group Decision Task Screen, they can see their transcript in the bottom half.
10. The meeting proceeds with participants having a conversation, trying to reach a consensus for the priority of items. During the meeting, the participants' voices will be recorded and used for near-real time statistics like time being silent, word count, etc.

11. The participant can leave the meeting by selecting “Leave Meeting”. Once they do so, they will see the survey page as shown in Figure 8.

- If the participant leaves the meeting by mistake, they can re-enter the meeting using the same meeting ID, given it is still in progress.

User Interface Screens

The user interface is divided into two groups of screens:

- Participant Pages
- Researcher/Admin Pages

Participant Pages:

- **Instructions Page:** The instructions page is the first page that the user encounters. It explains to the user how they will be able to participate in the research.

URL: <https://conversation-agent.herokuapp.com/>

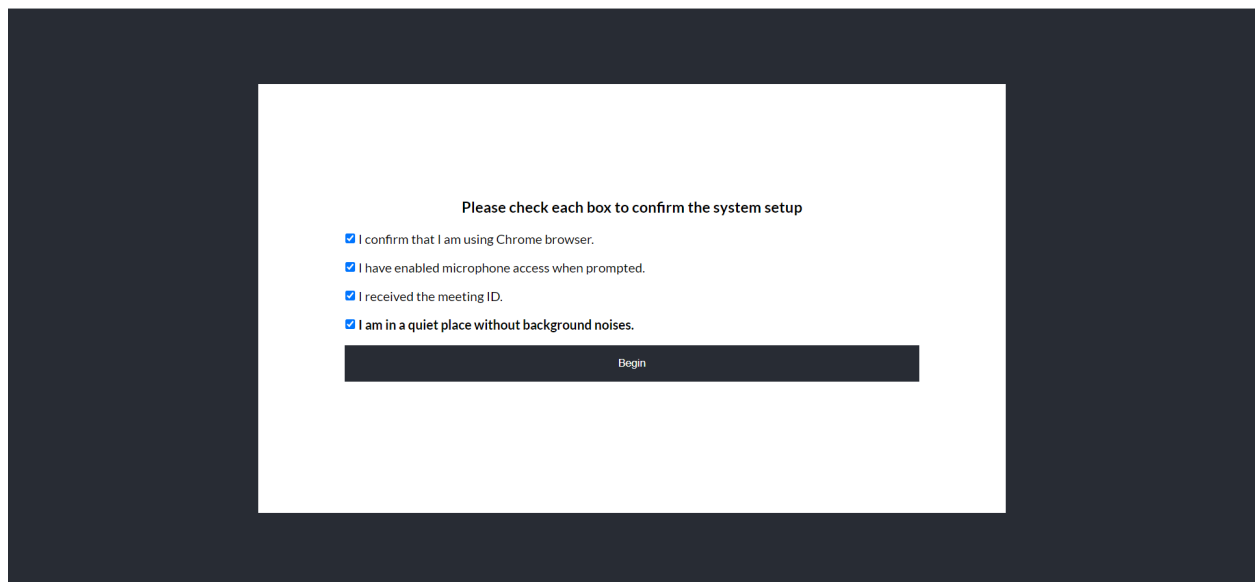


Figure 1: Instructions Page

- **Consent Page:** On this page, participants must provide consent to use their browser microphone, use their data, and observe their conversation.

URL: <https://conversation-agent.herokuapp.com/user-consent>

Purpose of Study
The goal of this study is to understand the group dynamics where multiple users verbally interact with each other and make a group decision through an online video conferencing platform. Participants will discuss with other participants to make a consensus on the item ranking and successfully submit the group decision.

Experiment Procedure
In this experiment, you will first be asked to make an individual decision on ranking the items for desert survival. Once submitting the result, you will join a group meeting and discuss with other participants to make a consensus on the item ranking. Once the group decision is made and submitted, a survey link will be provided.

Statement of Consent
☒ I understand and agree that my screen recording is necessary to participate in this study.
☒ I have read the above information. I consent to take part in the study.

Name
Net ID
Meeting ID

Next

Figure 2: Consent Page

- **Waiting Page:** The participant will be redirected to this page upon successful submission of the individual task. Once the researcher starts the group discussion, all the participants will be redirected from this page to the group decision-making task.

URL: <https://conversation-agent.herokuapp.com/waiting>

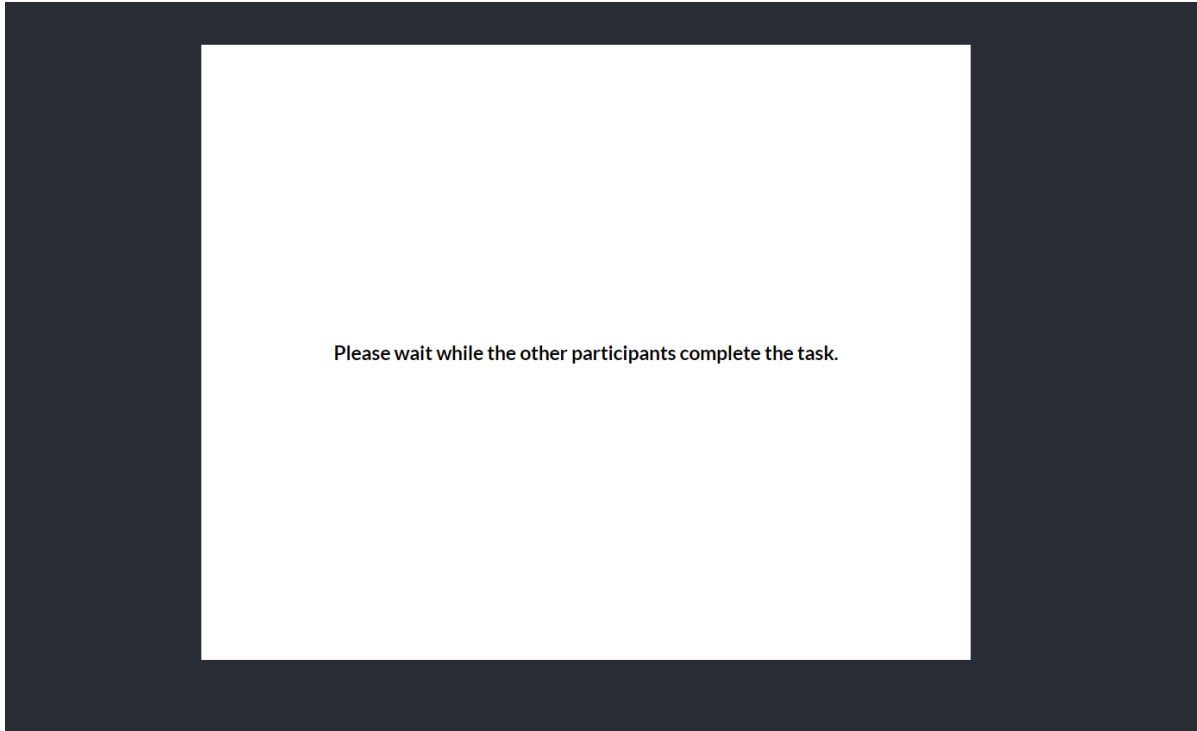


Figure 3: Waiting Page

- **Individual Task Page:** Once the participant fills out the consent form and enters the correct meeting ID, they will be redirected to this page where they can go through the prompt of the problem and drag and drop items to rearrange them according to the priority order.
URL: <https://conversation-agent.herokuapp.com/desert-problem>

Individual Decision-Making Task

Read the Desert Survival scenario and rank the items according to their importance to your survival in the desert.

Desert Survival

It is approximately 10am in mid-July and you have just crash landed in the Sonora Desert, near the Mexico-USA border. The plane has completely burnt out, only the frame remains. Miraculously, the 10 passengers are uninjured but the pilot has been killed.

The pilot was unable to tell anyone of your position before the crash. However, ground sightings taken shortly before the crash suggest that you are about 65 miles off the course filed in your flight plan. A few moments before the crash, the pilot indicated you were about 70 miles south east of a mining camp. The camp is the nearest known settlement.

The immediate area is quite flat and, except for the occasional thorn bush and cacti, is rather barren.

Before the plane caught fire, your group was able to save the 5 items on the desk.

Your task is to rank them according to their importance to your survival in the desert.

Drag and rank items from 1 to 5.

| | |
|---|--|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

| |
|----------|
| Knife 🔪 |
| Torch 🔦 |
| Pistol 🔫 |
| Water 💧 |
| Coat 🧥 |

Submit

Individual Decision-Making Task

Read the Desert Survival scenario and rank the items according to their importance to your survival in the desert.

Desert Survival

It is approximately 10am in mid-July and you have just crash landed in the Sonora Desert, near the Mexico-USA border. The plane has completely burnt out, only the frame remains. Miraculously, the 10 passengers are uninjured but the pilot has been killed.

The pilot was unable to tell anyone of your position before the crash. However, ground sightings taken shortly before the crash suggest that you are about 65 miles off the course filed in your flight plan. A few moments before the crash, the pilot indicated you were about 70 miles south east of a mining camp. The camp is the nearest known settlement.

The immediate area is quite flat and, except for the occasional thorn bush and cacti, is rather barren.

Before the plane caught fire, your group was able to save the 5 items on the desk.

Your task is to rank them according to their importance to your survival in the desert.

Drag and rank items from 1 to 5.

| | |
|---|----------|
| 1 | Knife 🔪 |
| 2 | Pistol 🔫 |
| 3 | Water 💧 |
| 4 | Coat 🧥 |
| 5 | Torch 🔦 |

| |
|--|
| |
| |
| |
| |
| |

Submit

Figures 4-5: Individual Task Page

- Group Task Page:** This page is pretty similar to the the Individual Page, the only difference being that on this page, all the users first have to discuss and then come to a consensus for the ranking list. The user can check their transcripts and also leave the meeting by scrolling down on this page. Once a user submits for the group, all other users will also be redirected to the survey page.

URL: <https://conversation-agent.herokuapp.com/client>

Group Decision-Making Task

Read the Desert Survival scenario and rank the items according to their importance to your survival in the desert.

Desert Survival

It is approximately 10am in mid-July and you have just crash landed in the Sonora Desert, near the Mexico-USA border. The plane has completely burnt out, only the frame remains. Miraculously, the 10 passengers are uninjured but the pilot has been killed.

The pilot was unable to tell anyone of your position before the crash. However, ground sightings taken shortly before the crash suggest that you are about 65 miles off the course filed in your flight plan. A few moments before the crash, the pilot indicated you were about 70 miles south east of a mining camp. The camp is the nearest known settlement.

The immediate area is quite flat and, except for the occasional thorn bush and cacti, is rather barren.

Before the plane caught fire, your group was able to save the 5 items on the desk.

Your task is to rank them according to their importance to your survival in the desert.

Drag and rank items from 1 to 5.

| | |
|---|--|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |

Knife 🔪

Torch 🔦

Pistol 🔫

Water 💧

Coat 🧥

Submit

Meeting ID: exte

Checking checking if the transcript is working.. That was great.. Parking passes for \$35. You know, seriously, that's pure bamboo wood.

Leave Meeting

Figures 6-7: Group Task Page

- Survey Page:** Once the meeting ends or the participant decides to leave the meeting, they will be redirected to the below page. At this point, their speech transcription and emotion is no longer being detected/processed.

URL: <https://conversation-agent.herokuapp.com/survey>

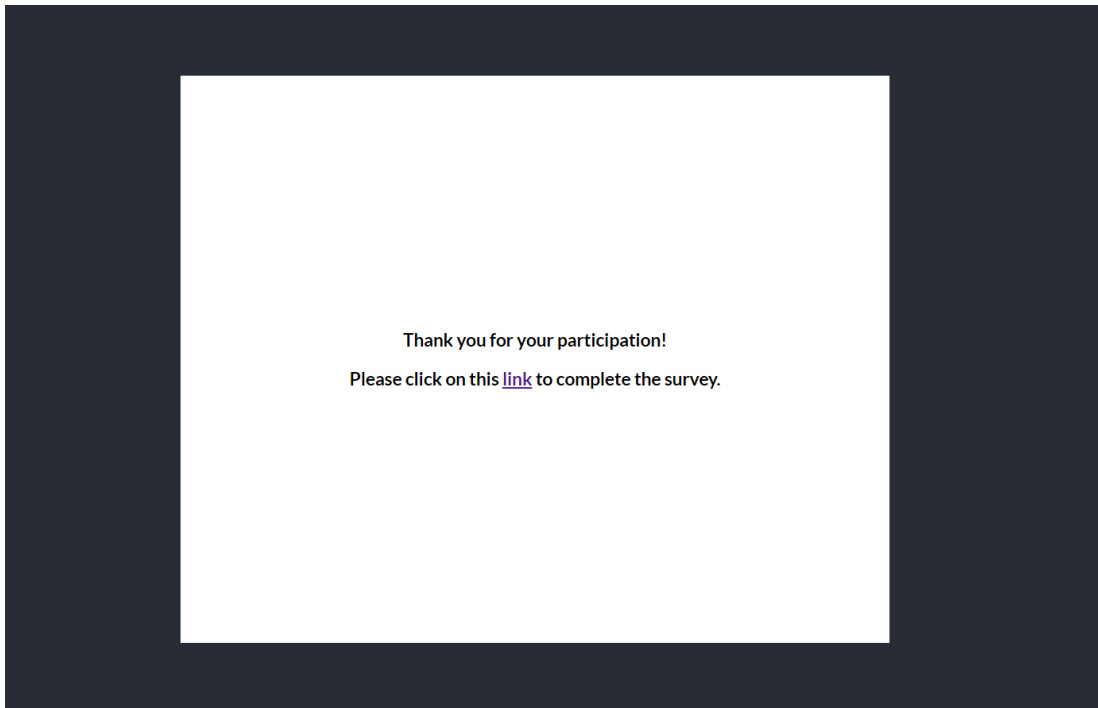


Figure 8: Survey Page

Researcher/Admin Pages:

- **Meeting Creation Page:** This page is the first page the researcher sees. Here, the researcher can create a meeting and share the meeting ID with the participants.

URL: <https://conversation-agent.herokuapp.com/create-meeting>

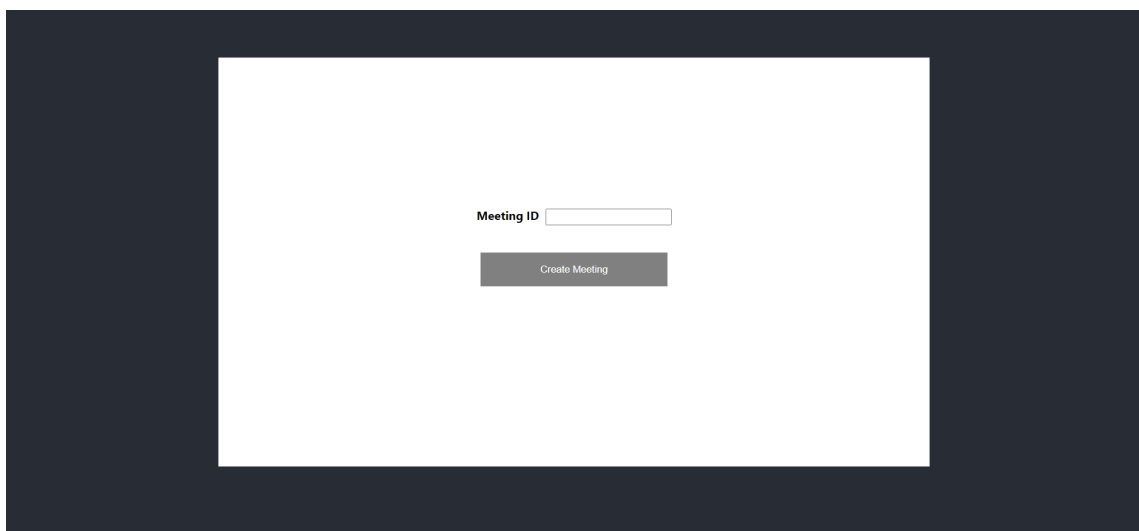


Figure 9: Meeting Creation Page

- **Main Meeting Page:** Once the meeting has been created, the researcher can use this page to view the existing participants, send personalized interjections to each participant, and end the meeting.

URL: <https://conversation-agent.herokuapp.com/admin>

Participants Submitted

Start Group Discussion

There are no records to display

Active Participants

Meeting ID: testmeeting1

Message:

Send To:

Send

End Meeting

Participants Submitted

Shreyas

0:5:38

| Users | Name | Word Count | Turn Taking | Time Silent |
|--------|---------|------------|-------------|-------------|
| ss3259 | Shreyas | 18 | 2 | 04:19 |

Active Participants

Shreyas

Meeting ID: exte

ss3259: Checking checking if the transcript is working.

ss3259: Parking passes for \$35. You know, seriously, that's pure bamboo wood.

Message:

Send To:

Send

End Meeting

Participants Submitted

Berk

0:0:36

| Users | Name | Word Count | Turn Taking | Time Silent |
|-------|------|------------|-------------|-------------|
| be236 | Berk | 0 | 0 | 00:07 |

Active Participants

Berk

Meeting ID: testmeeting11

be236: this red what's up 9 is it just needed to speak to this it's working now anyway I just have to talk to this thing to test it

Message:
Send To:
Send

End Meeting

Figures 10-12: Researcher's Main Meeting Page (with and without any participant's activity)

- Meeting End Page:** After the researcher has ended the meeting, they can view the entire summary and keywords of the meeting transcriptions. Same URL as above.

Participants Submitted

Berk

0:1:43

| Users | Name | Word Count | Turn Taking | Time Silent |
|-------|------|------------|-------------|-------------|
| be236 | Berk | 53 | 4 | 00:14 |

Meeting ID: testmeeting11 was ended by the admin.

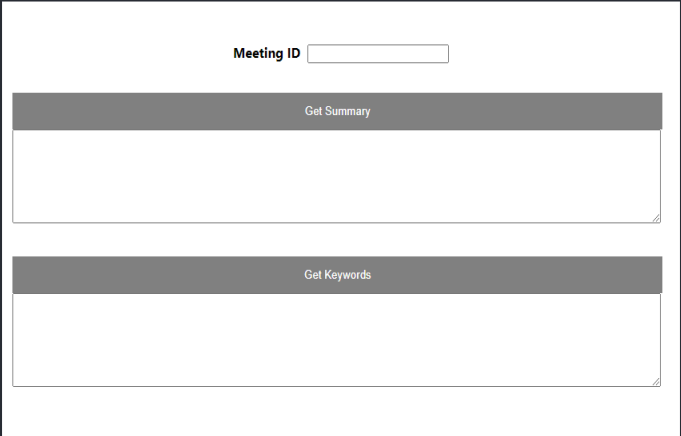
Get Summary

Get Keywords

Figure 13: Meeting End Page

- **Summary and Keywords Page:** If the researcher wants to fetch the summary of a specific meeting, they can get the summary and keywords of the meeting by entering the correct meeting ID.

URL: <https://conversation-agent.herokuapp.com/get-summary>



The screenshot shows a web interface with a dark background. At the top, there is a label "Meeting ID" followed by a text input field. Below this, there are two main sections. The first section has a grey header bar with the text "Get Summary". Below the header is a large white rectangular area, likely for displaying the meeting summary. The second section has a grey header bar with the text "Get Keywords". Below the header is another large white rectangular area, likely for displaying the meeting keywords. Both sections have a small icon in the bottom right corner of their white areas.

Figure 14: Summary and Keywords Page

Development

Database Structure

All relevant information is stored in the “data” database in MongoDB. All document stores are indexed by their respective meeting ID, aside from the “consent” collection. As shown in Figure 15, this collection simply stores the names and net IDs of participants who have consented to the study, along with the meeting ID of their meeting.

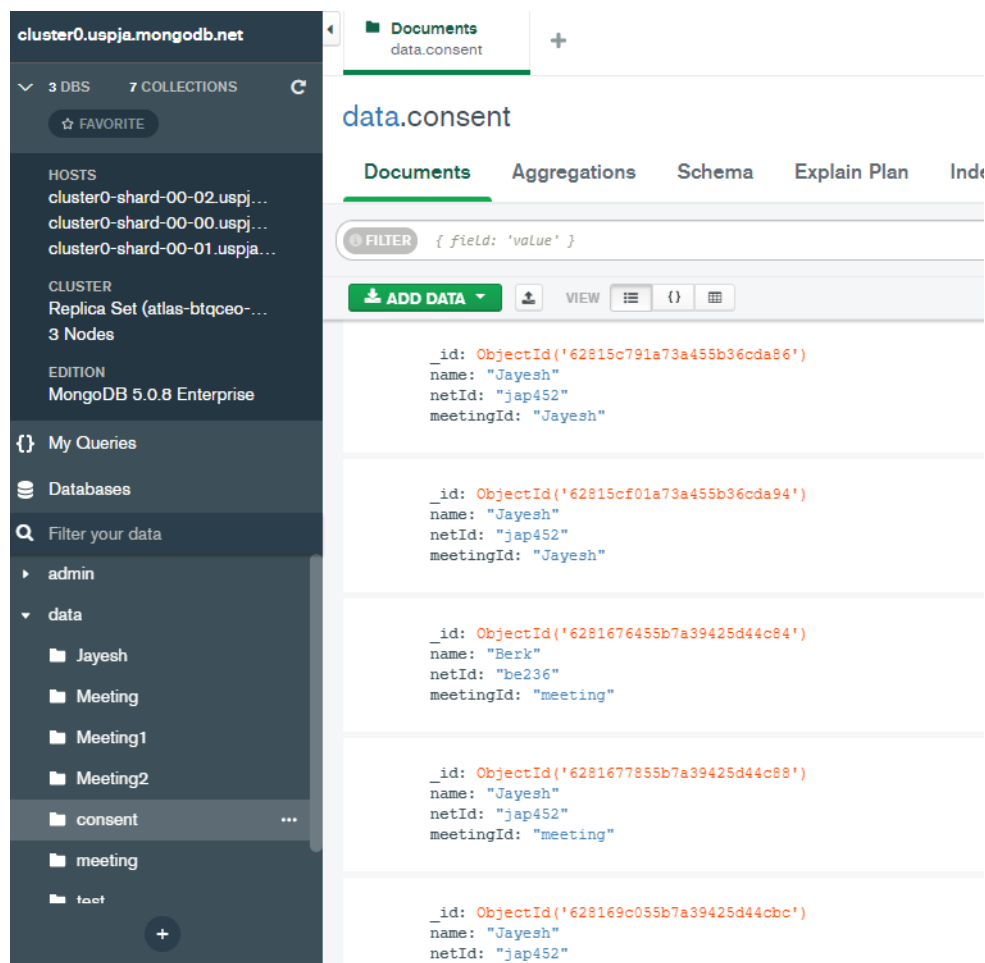


Figure 15: Consent Collection

Figure 16 displays the database collection for a meeting with meeting ID “final2”. The first document holds an “active” field. This field is set to “true” when a meeting is created and “false” when it ends.

Other documents are divided into documents of type “ping”, type “data”, type “choices”, and type “silent”. “Ping” documents store a net ID and ping count representative of how many times the program has pinged the participant to check that they are in the meeting. “Data” documents store conversation data with a corresponding net ID and timestamp. “Choices” documents store the order of items chosen in the individual and group task screens, and “Silent” documents store an integer representing the number of seconds a participant has been silent. “timeSilent” is tracked on the frontend. Every 7 seconds, we check whether a participant has spoken or not. If they have, time silent remains the same. If they have not, time silent is incremented by 7 and updated accordingly in the database.

| | |
|---|---|
|  | <pre>_id: ObjectId('638e51db8d6f265e5b7598ef') netId: "ss3259" > choices: Object timestamp: "2022-12-05T20:17:31.519Z" type: "choices"</pre> |
| | <pre>_id: ObjectId('638e51ec8d6f265e5b7598f1') netId: "sph75" > choices: Object timestamp: "2022-12-05T20:17:49.444Z" type: "choices"</pre> |
| | <pre>_id: ObjectId('638e52018d6f265e5b7598f2') netId: "avk57" timeSilent: 49 type: "silent"</pre> |
| | <pre>_id: ObjectId('638e520c8d6f265e5b7598f4') netId: "jk2345" timeSilent: 231 type: "silent"</pre> |
| | <pre>_id: ObjectId('638e52518d6f265e5b7598f7') netId: "mmc274" text: "so now you can see their synced up and now I'm going to ruin it now bu..." timestamp: "2022-12-05T20:19:29.521Z" type: "data"</pre> |
| | <pre>_id: ObjectId('638e52838d6f265e5b7598fa') netId: "mmc274" text: "there's a delay like from when I speak to when you see sweet to wait f..." timestamp: "2022-12-05T20:20:19.368Z" type: "data"</pre> |
| | <pre>_id: ObjectId('638e52978d6f265e5b7598fb') netId: "jk2345" text: "there's some delay compared to a job like more more Edison Light" timestamp: "2022-12-05T20:20:39.107Z" type: "data"</pre> |
| | <pre>_id: ObjectId('638e4e998d6f265e5b7598c0') type: "ping" netId: "ss3259" pingCount: 22</pre> |

Figure 16: Collection of meeting ID “final2”

API Calls

Our team has created two servers: one for serving REST APIs and the other for event publishing. The REST server is a Flask backend server that serves multiple admin and participant APIs. This service also talks to the MongoDB database to store data and to the IBM Cloud for tone classification and keyword extraction. The event publishing server is a [DeepStream.io](https://deepstream.io/) server that is used to send interjections from the researcher/admin screen to the individual participants.

API Details

| | Request & Response Body | Notes |
|--------------|--|--|
| User Consent | <p>Endpoint: POST: /userconsent</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Request:</p> <pre>{ "name": "name of user", "netId": "user's net ID", "meetingId": "meeting ID" }</pre> <p>Response:</p> <p>Success:</p> <pre>HTTP Code : 200 Body : { "success": true }</pre> <p>Failure (when meeting ID does not exist or is not active):</p> <pre>HTTP Code : 300 Body : {</pre> | <p>Participant API</p> <p>This API is used to capture user consent data.</p> <p>It first checks to see that the meeting exists and is active. If it isn't, it is unsuccessful.</p> <p>If these conditions are met, the participant's name, net ID, and corresponding meeting ID are stored in the collection containing all user consent information.</p> |

| | | |
|-------------------|--|--|
| | <pre>"success": false }</pre> | |
| Poll Conversation | <p>Endpoint: POST: /pollconversation</p> <p>Headers: 'Content-Type': 'application/json',</p> <p>Request:</p> <pre>{ "text": "user's speech converted to text", "netId": "user's net ID", "meetingId": "meeting ID", "timestamp": "UTC-timestamp" }</pre> <p>Response:</p> <p>Success:</p> <pre>HTTP Code : 200 Body : { "emotions": { "excited": 0.6, "frustrated": 0, "impolite": 0.4, "polite": 0, "sad": 0, "satisfied": 0, "sympathetic": 0 } }</pre> <p>Failure (due to text having < 4 words):</p> <pre>HTTP Code : 204 Body : { "emotions": { "excited": 0, "frustrated": 0, "impolite": 0, "polite": 0, "sad": 0,</pre> | <p>Participant API</p> <p>This API is called every 7 seconds from the UI and is used to send the user's spoken words as text to the backend.</p> <p>The IBM API requires at least 4 words to extract emotions; when "text" has fewer than four words, zeros are hardcoded into the response to prevent errors.</p> <p>When "text" contains 4 or more words, this text is stored in the database in a new, timestamped entry. Furthermore, the IBM API is called, and the extracted emotions are returned in the response.</p> |

| | | |
|--------|---|---|
| | <pre> "satisfied": 0, "sympathetic": 0 } </pre> | |
| Finish | <p>Endpoint: POST: /finish</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Request:</p> <pre> { "meetingId": "meeting ID", "netId": "user's net ID" } </pre> <p>Response:</p> <pre> HTTP Code : 200 Body : { "success": true } </pre> | <p>Participant API</p> <p>This API is called by the participant at the end of the meeting.</p> <p>This internally deletes the participant's ping count entry in the backend, which will hence remove them from the active participants list that is visible to the researcher.</p> |

| | | |
|-------------------------------|--|--|
| <p>Submit Ranking Choices</p> | <p>Endpoint: POST: /submitChoices</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Request:</p> <pre>{ "choices": ["choice 1", "choice 2", "choice 3"], "meetingId": "meeting ID", "netId": "user's net ID", "timestamp": "UTC-timestamp", }</pre> <p>Response:</p> <pre>HTTP Code : 200 Body : { "success": true }</pre> | <p>Participant API</p> <p>This API is called by the participant to submit their individual rankings.</p> <p>A data entry of type "choices" is stored in the meeting's collection with the submitting user's net ID.</p> |
|-------------------------------|--|--|

| | | |
|----------------------|---|--|
| Increment Ping Count | <p>Endpoint: POST: /incrementPingCount</p> <p>Headers: 'Content-Type': 'application/json',</p> <p>Request:</p> <pre>{ "netId": "user's net ID", "meetingId": "meeting ID", }</pre> <p>Response:</p> <pre>HTTP Code : 200 Body : { "success": true }</pre> | <p>Participant API</p> <p>This API is called every 7 seconds from the UI and is used to increment the participant's ping count at a fixed rate.</p> <p>If the participant already has a ping count, it gets incremented. If they do not but another participant does, the new ping count is set to the existing ping count. If no participants have a ping count yet, the new ping count is set to 1.</p> <p>These mechanisms ensure that all active participants will have synchronized ping counts, which is used to detect dropped participants.</p> |
|----------------------|---|--|

| | | |
|-----------------|---|--|
| Set Time Silent | <p>Endpoint: POST: /setTimeSilent</p> <p>Headers: 'Content-Type': 'application/json',</p> <p>Request:</p> <pre>{ "newTimeSilent": 14, "netId": "user's net ID", "meetingId": "meeting ID", }</pre> <p>Response:</p> <pre>HTTP Code : 200 Body : { "success": true }</pre> | <p>Participant API</p> <p>This API is called from the UI any time a user's time silent increases. It is used to increment the time silent in the database.</p> <p>The time silent is tracked on the frontend. Every 7 seconds, we check to see if speech has been detected. If no speech has been detected, the time silent is incremented by 7 seconds, and the new time silent is stored using this API. If speech has been detected, the time silent stays the same and the API is not called.</p> |
|-----------------|---|--|

| | | |
|---------------------------|--|--|
| <p>Create Meeting</p> | <p>Endpoint: POST: /createMeeting</p> <p>Headers: 'Content-Type': 'application/json',</p> <p>Request:</p> <pre>{ "meetingId": "meeting ID" }</pre> <p>Response:</p> <p>Success:</p> <pre>HTTP Code : 200 Body : { "success": true }</pre> <p>Failure (when meeting ID is already in use):</p> <pre>HTTP Code : 300 Body : { "success": false }</pre> | <p>Admin API</p> <p>This API is used at the start of the experiment by the researcher to create a meeting with the specific meeting ID.</p> <p>This API first searches the list of existing meetings. If there is a match, it will return a failure response, which signifies that the admin has to choose a different meeting ID.</p> <p>If the meeting ID has not been used for earlier meetings, then a collection is created in MongoDB and a success response is returned.</p> |
| <p>Meeting Transcript</p> | <p>Endpoint: GET: /transcript?meetingId=1234</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Response:</p> <pre>HTTP Code: 200 { "transcript": "meeting transcript", }</pre> | <p>Admin API</p> <p>This API returns the meeting transcript.</p> <p>It scans through the meeting's entire database collection, concatenating all of the text in a chronological order. The text is also annotated with the user that said it.</p> |

| | | |
|---------------------|--|---|
| Active participants | <p>Endpoint: GET: /activeParticipants?meetingId=1234</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Response:</p> <pre> HTTP Code : 200 { "netId1": "name1", "netId2": "name2" } </pre> | <p>Admin API</p> <p>This API returns names and net IDs of active participants in the meeting.</p> <p>This is used by the researcher to make sure the participants are all on the call and no one drops.</p> <p>The active participant list is maintained using ping counts incremented by the incrementPingCount API.</p> <p>If a user drops from the meeting (intentionally or due to technical errors), then incrementPingCount will no longer be triggered from the UI. The system detects that their ping count is not updating, so we remove that particular user's name and netId from the set of active participants.</p> |
| End Meeting | <p>Endpoint: POST: /endMeeting</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Request:</p> <pre> { "meetingId": "meeting ID" } </pre> <p>Response:</p> <pre> HTTP Code : 200 </pre> | <p>Admin API</p> <p>This API is used by the researcher to end the meeting and mark it as inactive in MongoDB.</p> |

| | | |
|-----------------|--|---|
| | <p>Body :</p> <pre>{ "success": true }</pre> | |
| Meeting Summary | <p>Endpoint: POST: /summary</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Request:</p> <pre>{ "meetingId": "meeting ID" }</pre> <p>Response:</p> <p>Success:</p> <pre>HTTP Code : 200 Body: { "summary" : "Summary of the conversation" }</pre> <p>Failure (when meeting ID does not exist):</p> <pre>HTTP Code : 404 Body: { "summary": "Meeting not found" }</pre> | <p>Admin API</p> <p>This API call gets the meeting summary after the end of the meeting.</p> |

| | | |
|------------------------|---|--|
| Meeting Keywords | <p>Endpoint: POST: /keywords</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Request:</p> <pre>{ "meetingId": "meeting ID" }</pre> <p>Response:</p> <p>Success:</p> <pre>HTTP Code: 200 Body: { "keywords" : ["KW1", "KW2"] }</pre> <p>Failure (when meeting ID does not exist):</p> <pre>HTTP Code : 404 Body: { "keywords": "Meeting not found" }</pre> | <p>Admin API</p> <p>This API call fetches the keywords of the meeting/conversation after it ends.</p> <p>We are using an external service from IBM Cloud keyword extraction.</p> |
| Submitted Participants | <p>Endpoint: GET: /submittedParticipants?meetingId=1234</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Response:</p> <pre>HTTP Code : 200 { "netId1": "name1", "netId2": "name2" }</pre> | <p>Admin API</p> <p>This API returns names and net IDs of participants in the meeting that have submitted their ranking choices.</p> <p>This is used by the researcher to make sure the group discussion is not started until everyone has submitted.</p> <p>The meeting's collection is queried to find all data</p> |

| | | |
|--------------------|---|---|
| | | <p>entries of type “choices”. This provides the net IDs of the users that submitted. These net IDs are then mapped to the users’ names using the data in the consent collection.</p> |
| Participant Counts | <p>Endpoint: GET: /participantCounts?meetingId=1234</p> <p>Headers: 'Content-Type': 'application/json'</p> <p>Response:</p> <pre> HTTP Code : 200 { "wordCounts": { "netId1": 3, "netId2": 28 }, "turnCounts": { "netId1": 1, "netId2": 3 }, "timeSilent": { "netId1": "01:00", "netId2": "00:15" }, "names": { "netId1": "name1", "netId2": "name2" } } </pre> | <p>Admin API</p> <p>This API returns all live participant data for every user in the meeting. This includes: word counts, turn counts, time silent, and their names. This is used to display the data to the researcher live in a table.</p> <p>Names are retrieved similar to before using the consent collection.</p> <p>Word counts are computed by summing the number of words stored in each participant’s data entries.</p> <p>Turn counts are computed by summing the number of individual data entries a user has.</p> <p>Time silent is already stored in seconds, and this is converted into a string in “mm:ss” format.</p> |

Environment Setup

Frontend Setup

The frontend codebase can be found at the following Github repository's **feature-branch**: <https://github.com/JieunKimHCI/convo-FE>

For local setup, the following must be installed:

- Node.js (latest version) added to the PATH
- Deepstream.io
- Git

Frontend Installation Steps:

- **Note about Deepstream:**
 - The frontend code is configured to use the Deepstream server deployed on Heroku (see deployment section).
- ```
const { DeepstreamClient } = window.DeepstreamClient;
const client = new DeepstreamClient('wss://conversation-agent-deepstream.herokuapp.com');
client.login();
```
- To use a local server instead, execute “./deepstream.exe start” and change all client urls to “localhost:6020”.
  - Clone the repository by executing:
    - “git clone <https://github.com/JieunKimHCI/convo-FE>”
  - Navigate to the project directory
  - Run “npm install” to install the dependencies from “package.json”
  - Run “npm start” to start the server - it runs on port 3000 by default

## Backend Setup

### REST Server

**Github Repository:** <https://github.com/JieunKimHCI/convo-BE>

This repository contains the backend code base with instructions on setting up the backend codebase, connecting to MongoDB, and running the program locally.

**.env Config File:**

The config file contains credentials for the MongoDB endpoint, passkeys, and IBM cloud endpoint and credentials. It should look as follows:

```
MONGO_USERNAME = "dev"
MONGO_PW = "conversational_agent"
MONGO_CLUSTER = "cluster0"
IBM_APIKEY = "<your IBM ABI KEY>"
IBM_URL= "your IBM URL>"
```

### ***requirements.txt***

This file consists of Python library dependencies (required on local installation and Heroku)

### ***nltk.txt***

This file lists the NLTK dependencies (required on local installation and Heroku)

## **Event Publishing Server**

**GitHub Repository:** <https://github.com/JieunKimHCI/convo-deepstream>

This repository contains the code forked from Deepstream's git repository. We have done some minor tweaks to it so that we can deploy it as a standalone Node.js server on Heroku. No other change is required from the backend.

## **Database Setup**

To gain read/write access to MongoDB, you must be invited to the Conversational Agent project by an administrator (Jieun Kim). This can be done [here](#). New users must have a MongoDB account.

## **IBM Cloud Setup**

1. Create an IBM Cloud account [here](#).
2. Create a Basic subscription (you will need to enter a credit/debit card).
3. Create a resource [here](#) and copy the Access Key.
4. For local deployments enter the access key in the ".env" config file.
5. To add it to Heroku deployment, refer to the deployment section.

# Deployment

Listed below are the steps required to deploy both the REST Server and the Deepstream server to Heroku.

## ***Packaging Frontend with Backend:***

1. Create the frontend build from the frontend codebase (“npm run build”).
2. Copy the “build” folder and paste it into the backend code root folder.

## ***Deploying REST Server:***

1. Create a Heroku account [here](#).
2. Create a new app (<https://dashboard.heroku.com/new-app>). **You must configure the app to use Eco dynos in the “Resources” tab.** This requires either linking a credit card or using GitHub Student credits.
3. Setup Heroku Git [here](#).
4. Add the config vars in the “Settings” tab under the “Config Vars” section. These are the parameters in the “.env” local file.
5. Once you have created a new UI build and replaced the build folder in the backend code, push to Heroku by running the following from the backend directory:
  - a. heroku login
  - b. heroku git:remote -a <app-name-on-heroku>
  - c. git push heroku main **OR** git push heroku <branch name>:main
6. The app is now deployed.

Config Vars Hide Config Vars

|                |                                        |                                                                                                                                                                             |
|----------------|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IBM_APIKEY     | r8HFtmngPxLLAPZbvqNcTlye5yteWtbTb3XhCI |   |
| IBM_URL        | https://api.us-east.natural-language-t |   |
| MONGO_CLUSTER  | cluster0                               |   |
| MONGO_PW       | conversational_agent                   |   |
| MONGO_USERNAME | dev                                    |   |
| KEY            | VALUE                                  |                                                                                        |









### **Deploying Deepstream Server:**

1. Follow steps 1-3 above.
2. Add the below configuration parameters in the “Settings” tab under the “Config Vars” section.
3. Pull down <https://github.com/JieunKimHCI/convo-deepstream>.
4. From the convo-deepstream directory, run the following:
  - a. heroku login
  - b. heroku git:remote -a <app-name-on-heroku>
  - c. git push heroku main
7. The app is now deployed.

Config Vars

Hide Config Vars

|                       |       |                                                                                                                                                                         |
|-----------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NPM_CONFIG_PRODUCTION | false |   |
| PORT                  | 6020  |   |
| YARN_PRODUCTION       | false |   |
| KEY                   | VALUE | Add                                                                                                                                                                     |