

EE 547 – Applied and Cloud Computing

Project Report

Wenhao Chi, Haoyu Xie

May 7, 2023

Project Title: Multifunctional Customer Service Platform

Summary and Description

Our objective in this project is to create a customer service platform with live chat capabilities that enable seamless communication between the customer and the customer service representative, as well as between the representative and their colleagues through text and voice messages. We have developed the platform with accessibility in mind, taking into consideration the needs of users who may find typing or reading messages inconvenient. To address this, our app is equipped with voice-to-text conversion technology to ensure that all communications are recorded in text format, allowing agents to process information efficiently, while replies can be sent to users via phone voice or text.

To guarantee the security and privacy of our users, the platform uses token-based authentication. Moreover, customers who do not wish to register can still communicate with customer service agents, enhancing ease of use and accessibility. Our application's adaptable framework allows for the seamless integration of additional functionality, such as dial robot, to meet the evolving needs of the platform and its users.

1 Proposed Architecture

Our project comprises a web-based frontend and a backend server, which communicate via a REST API. The backend server, built on Node.js, serves a variety of purposes, including but not limited to saving, updating, and retrieving data from the MongoDB database. It also processes and computes data to meet the requests of the front-end, exchanging information with the front-end via designated endpoints.

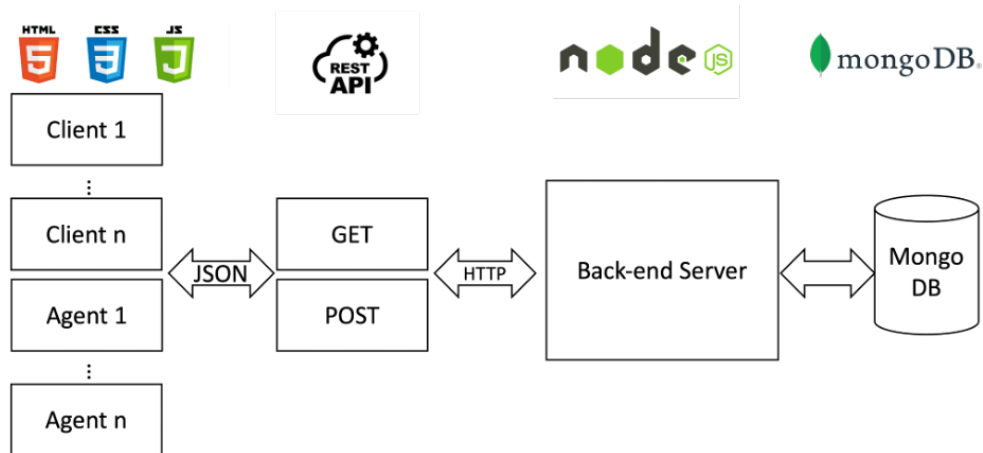


Figure 1: Overall Application Workflow

The front-end, developed using HTML and JavaScript, is responsible for rendering the user interface and ensuring a smooth and intuitive experience for users engaging with the platform. Users interact with the RESTful API through a dedicated front-end client, which may take the form of a Windows application or similar. A visual representation of the overall project architecture is depicted in Fig. 1 below.

To bring this application to fruition, two crucial resources are necessary: the Web Speech API and a CSS template. The Web Speech API plays a fundamental role in facilitating voice-to-text and text-to-voice conversion, enabling real-time transcription of voice messages and their subsequent transformation into written text. This functionality is especially critical for users who may find typing messages challenging or prefer to speak their thoughts aloud. The CSS template, on the other hand, is instrumental in enhancing the application's overall presentation and aesthetics. By incorporating a visually appealing template, the user interface becomes more engaging and intuitive, increasing the likelihood of user satisfaction and adoption. Therefore, the combination of the Web Speech API and the CSS template is essential in creating an effective and user-friendly platform.

Overall Application Workflow: As shown in Fig. 1, the chat application's architecture incorporates a variety of technologies to ensure seamless operation. On the left side, HTML, CSS, and JavaScript are employed to develop the front-end, which communicates with the backend using RESTful APIs. Upon receiving requests from the front-end, the backend performs the necessary actions, such as creating, updating, or finding data within MongoDB based on the frontend's requirements.

MongoDB serves as the platform's storage solution for all vital information, including messages, sessions, and personnel data. When clients request new messages, the backend server retrieves the relevant data via the RESTful API, extracting the necessary information from MongoDB and delivering it to the requesting client in a timely and efficient manner. This intricate workflow enables effective management of data and ensures a seamless and user-friendly experience across the platform, ultimately enhancing user satisfaction and

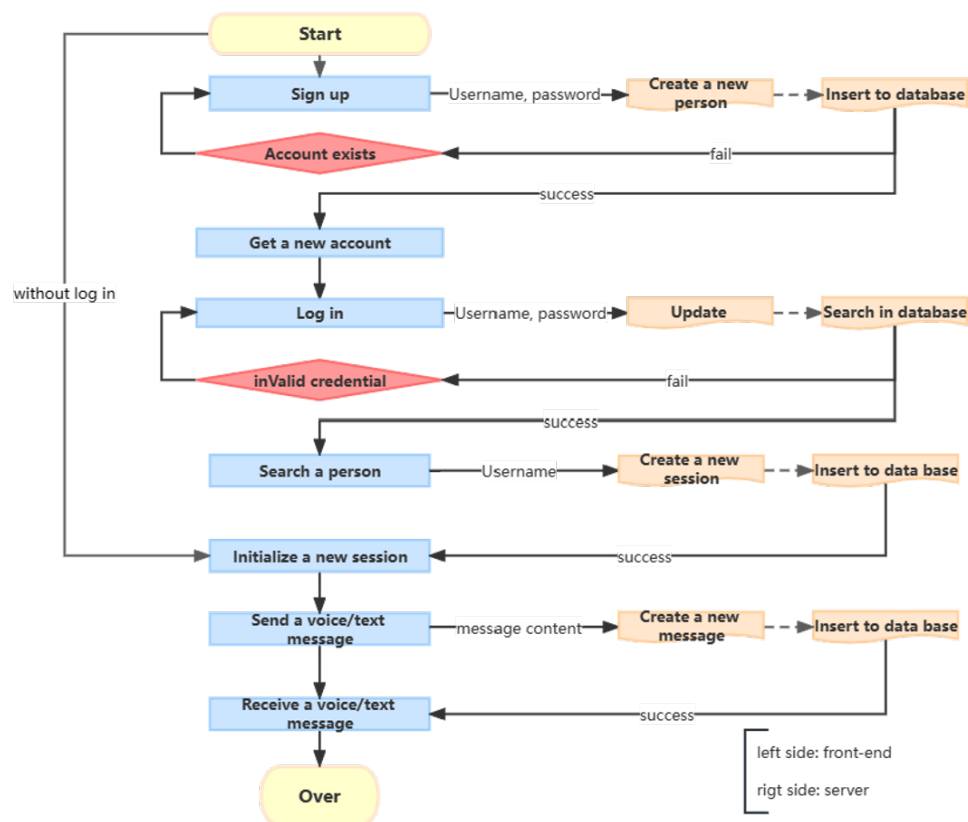


Figure 2: Overall Logic Flow

engagement.

Overall Logic Flow: As shown in Fig. 2 below,

- Users sign up:
 - a. Check for account existence.
 - b. If the account does not exist, create a new account.
- Users log in:
 - a. Check for credential validation.
 - b. If credentials are valid, grant access to the user.
- Users search for another user to launch a session.
- Users send voice and text messages within the session.
- Users receive voice and text messages from other participants in the session.

Overall Data Flow: As shown in Fig. 3 and Fig. 4 below,

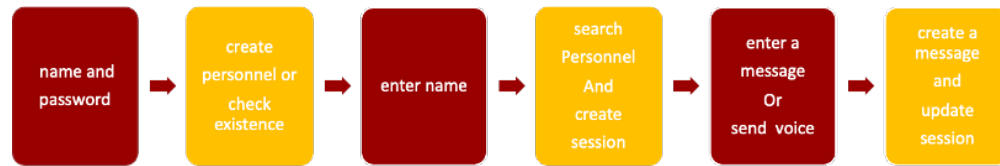


Figure 3: From front-end to server

- From front-end to server:
- From server to front-end:

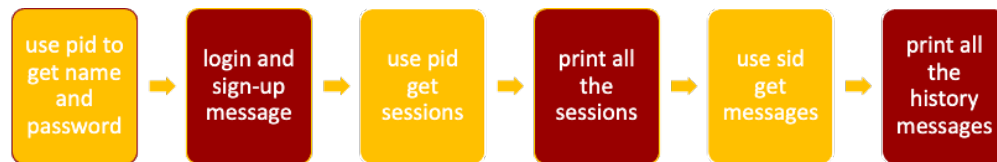


Figure 4: From server to front-end

We will use a local mySQL database server to store user data and likely AWS S3 to store the original and processed files in the cloud. This will simplify the async audio processing pipeline.

The application will transcode and perform audio processing *off-line*. We will either use a task server to process content and report to the backend or investigate using Amazon Elastic Transcoder but pricing may be prohibitive.

2 Implementation

Frontend - Web Application

Our web-based frontend is developed using HTML, CSS, and JavaScript, offering a more straightforward development process and increased flexibility compared to a Windows application. With access to various open-source tools, enhancing the user interface and overall aesthetics becomes a breeze. This application contains multiple frontend pages supporting a range of features, including sign-up, log-in, initiating new sessions, switching between sessions, sending voice and text messages, and browsing chat history.

The following is a detailed introduction of the front pages:

Log In Page: Log in page as shown in Fig. 5.

- Login to the account by providing the account name and password.

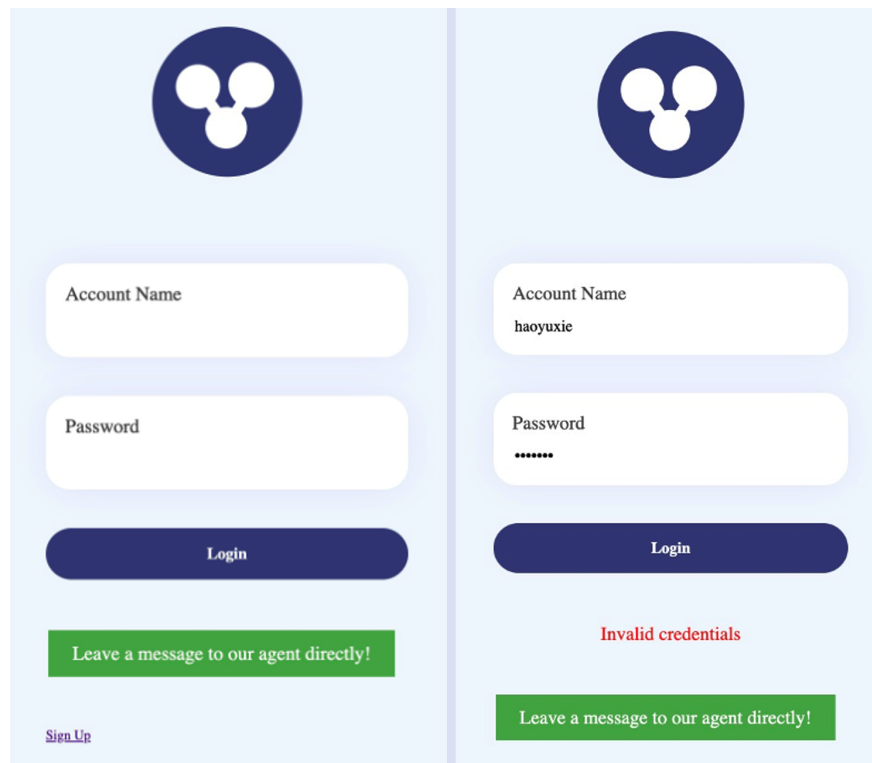


Figure 5: From server to front-end

- Alternatively, send a voice message without undergoing the login process.
- Sign up for a new account to gain access to additional features.
- Authentication is unsuccessful if the password entered is invalid.

Sign Up Page: sign up page as shown in Fig. 6.

- Enter the account name and password to commence the sign-up process.
- Select your role as either an "agent" or "customer".
- Click the "Leave a message to our agent directly" option to send a voice message without undergoing the login process.
- Sign in using your existing account credentials if you already have an account.

Voice and Text Message Page: Voice and Text Message page as shown in Fig. 7.

- Initiate the recording process by clicking the "Record" button.
- End the recording process by clicking the "Stop" button.
- Display the recorded voice message in the designated box.
- Showcase the transcript of the recorded voice message in the text box

Figure 6: From server to front-end

- Send the voice message, accompanying text, name, and phone number to an active agent by clicking the "Send" button
- Provide feedback (as shown in Fig. 8) on the sending result after clicking the "Sending" button.

Chat Page: Login page as shown in Fig. 9.

- Utilize the search box located at the upper left-hand corner of the interface to search for and locate individuals, and initiate a session with them.
- Access different sessions by clicking on the session listing displayed on the left-hand side of the interface.
- Enter text into the message input field, and send it by clicking the "send" button located at the bottom of the interface.
- Display the chat history upon opening a session, providing users with a comprehensive overview of previous interactions.

Backend - MongoDB, RESTful API The backend server is responsible for communicating with the frontend through a RESTful API, ensuring seamless data exchange and

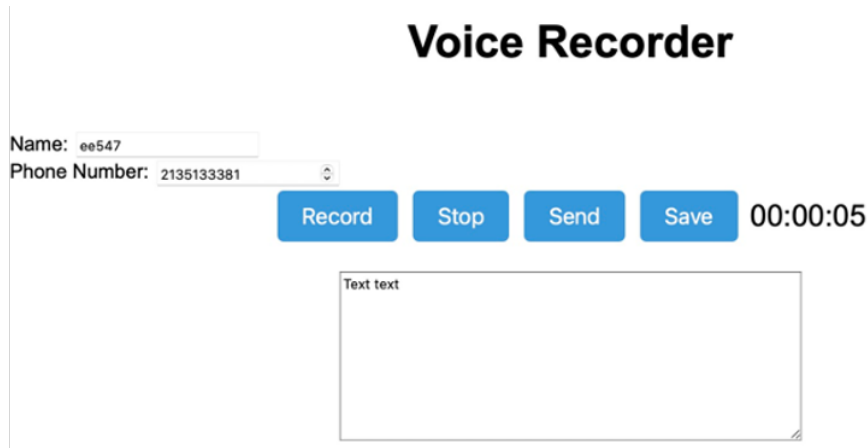
A web interface titled "Voice Recorder". It features a form with two input fields: "Name:" containing "ee547" and "Phone Number:" containing "2135133381". Below these fields are four blue buttons: "Record", "Stop", "Send", and "Save". To the right of the buttons is a digital timer showing "00:00:05". Below the buttons is a large text area with the placeholder text "Text text".

Figure 7: From server to front-end

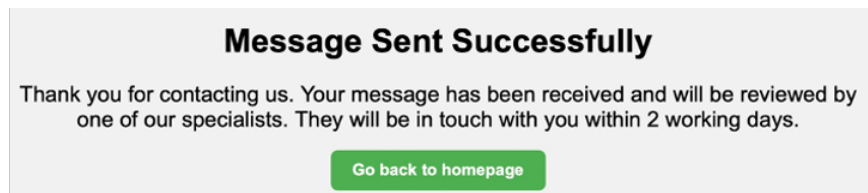
A light gray rectangular box containing a success message. At the top, it says "Message Sent Successfully" in bold. Below that, it says "Thank you for contacting us. Your message has been received and will be reviewed by one of our specialists. They will be in touch with you within 2 working days." At the bottom, there is a green button with the text "Go back to homepage".

Figure 8: From server to front-end

access to information related to users, sessions, and messages as required by the front-end. Additionally, the backend server interacts with MongoDB to perform various data operations such as creating, updating, searching, and retrieving information on individuals, sessions, or messages. This robust infrastructure ensures efficient data management and seamless user experience across the platform.

RESTful API: the communication between the frontend and backend is facilitated through a series of RESTful APIs, designed to handle data related to individuals, sessions, and messages. These APIs adhere to the "No Verb" rule, ensuring a consistent and standardized approach to data exchange, while promoting simplicity and maintainability of the platform's architecture.

- GET /ping: check if the server is running. It should return a simple "pong" response.
- GET /income: render a web page for sending a voice message.
- GET /succ: render a web page indicating that a message has been successfully sent.
- GET /login: render a web page for logging in to the application.
- GET /sign-up: render a web page for signing up for the application.
- GET /protected: return the username of the currently logged in user. This endpoint is protected and requires authentication.

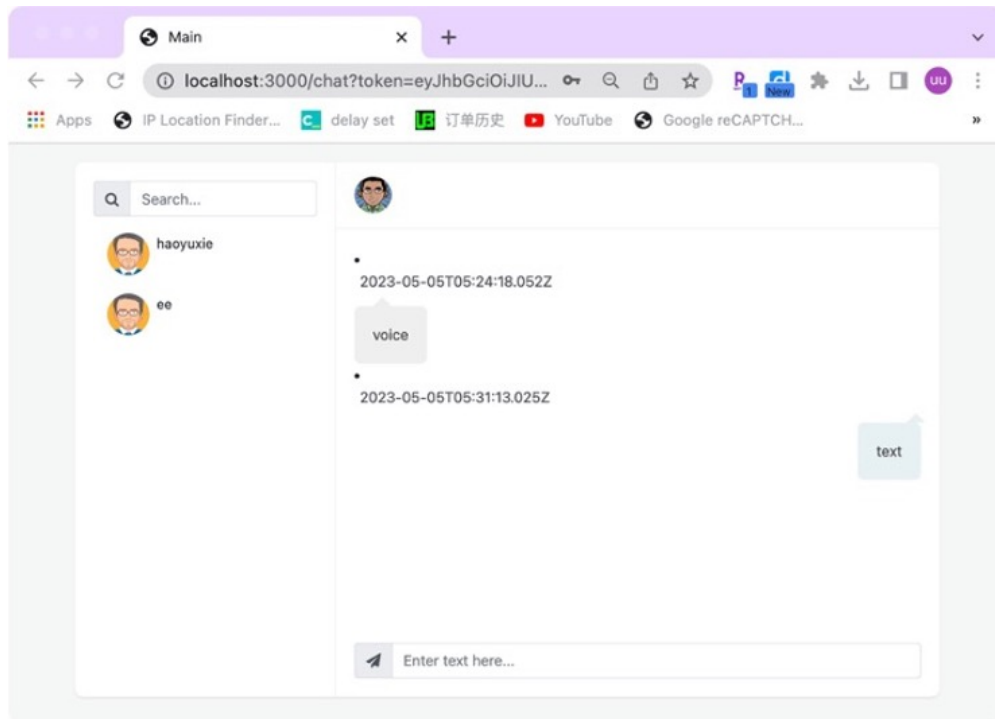


Figure 9: From server to front-end

- GET /personnel/:pid: return a list of messages associated with a particular person identified by person's ID.
- POST /signup: submit sign-up information for a new user. It should return a response indicating whether the sign-up was successful.
- POST /login: submit login credentials for an existing user. It should return a response containing an authentication token that can be used to access protected endpoints.
- POST /voice: submit a voice message and convert it to text. It should return the text of the message.
- POST /session: submit a message and the names of the people it is associated with. It should return a response indicating whether the message was successfully sent.

Database MongoDB is utilized to store all user-generated data in the platform. The database is composed of three collections: Personnel, Session, and Message. We have implemented the database using a MongoDB image from Docker Hub. MongoDB was chosen for this project due to its atomicity during write operations, which is crucial as the majority of server requests involve logging user and group messages. The database is read only when a user logs in, a new user signs up, a new session is initialized, or messages are requested. Here is a detailed description of the MongoDB collections. While many share similarities, they differ in terms of ID and data types:

Personnel Data Struction

Name	Type	Content
_id	ObjectId	User ID
username	string	Username
password	string	Password
created_at	Date	User Created Date
role	enum: “Customer” “Agent”	User Role
sessions	[sid] (List of session.id)	Seesions linked to this user

Session Data Struction

Name	Type	Content
_id	ObjectId!	Session ID
p1	ObjectId	ID of User 1
p2	ObjectId	ID of User 2
created_at	Date!	Session Created Date
last_update	Date	Last Update Date
messages	[mid] (List of message.id)	Messages linked to this session

Message Data Struction

Name	Type	Content
_id	ObjectId!	Message ID
pid	ObjectId	ID of User who sent this message
sid	ObjectId	ID of Session where this message be sent
content	String	Message content

3 Consumer/client/end-user experience

Ideally, customers should be able to seamlessly initiate communication with agents by sending text and voice messages, without being required to undergo any tedious sign-up process. This will enable a frictionless exchange of messages, with customers automatically being allocated accounts upon sending messages. Furthermore, an efficient system should be in place for customers to download important voice files and text messages to their local devices, ensuring seamless archiving.

Once users have created an account, the application should provide a comprehensive list of active chats, complete with the names of participants. This feature should enable users to effortlessly switch between chats and effectively initiate communication with target participants. Such functionality should guarantee an optimal and efficient communication experience.

4 Planned (incomplete/unimplemented features)

Incomplete Features:

- Video chat
- Video/image messages
- Group chats
- Forward messages to other session
- Auto-refresh of message content (web-socket based listening)

5 Conclusion

This project showcases the development of an accessible and user-friendly online chat platform that enables seamless communication between customers and agents. The platform integrates both text and voice messaging capabilities, catering to users with diverse communication preferences and abilities. A web-based frontend and a robust backend server ensure efficient data management and a smooth user experience.

The platform's security is enhanced through encryption and token-based authentication. Additionally, the platform's flexible framework enables the integration of additional features, including content moderation, to accommodate the evolving needs of its users. With its intuitive user interface, streamlined account creation process, and the ability to send messages without signing up, this chat platform caters to a diverse user base.

The successful completion of this project highlights the significance of accessible and secure communication tools in today's interconnected world. It serves as a foundation for further improvements and feature expansion to better meet the needs of its growing user base.

6 Team Roles

The following is the final breakdown of roles and responsibilities for our team:

- Wenhao Chi: Primary: Endpoints functions and its helper functions, front-end java script functions, front-end, debug. Secondary: database operation functions..
- Haoyu Xie: Primary: database operation functions, UI designing. Secondary: End-points functions, front-end java script functions.

All team members work on the final presentation, slides, and report.

7 References

- **Speech API:** <https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition>
- **UI Template:** <https://www.bootdey.com/snippets/tagged/chat>
- **MongoDB Node:** <https://www.mongodb.com/docs/drivers/node/current/>