

Software Engineering department

Braude College

Capstone project phase B – 61998

Video Conferencing Application with Speech Transcription

24-1-D-19

Mentor: Ronen Zelber

Students: Rami Amasha, Moayed Hamze

Github Link: <https://github.com/RamiAmasha31/rm-video-call>

1. Abstract.....	4
2. Introduction	5
3. Solution Description.	6
3.1. General Description.	6
3.2. Architecture Overview.....	7
3.2. Activity Diagram.....	9
3.3. Sequence Diagram.....	10
4. Development Process.....	11
4.1. What We Did to Develop and Build the System.	11
4.2. Tools Used.....	12
4.3. Client interaction during development.	12
4.4. Challenges Encountered and How We Addressed Them.	13
4.5. Decisions Made.	13
5. Verification and Evaluation.	14
5.1. Testing Process.....	15
5.2. Testing Results	16
6. Outcomes and Conclusions.....	30
6.1. Project Goals: Explanation and Justification.	30
6.2. Addressing challenges.....	31
6.3. Decision making considerations.....	31
7. Lessons learned.....	32
7.1. Success criteria.	32
8. Conclusions.	33
9. User Guide.	34
10. Maintenance guide.....	41
10.1. Purpose.....	41
10.2. System environment.....	41
10.3. Software installation and setup.....	42
10.4. Documentation.....	44
11. Appendix.....	47

1: System Architecture	7
2: Activity Diagram	9
3: Sequence Diagram.....	10
4: Page Load Test Result	16
5: Logs Screen	18
6: Meeting Room	19
7: Join Meeting Dialog	20
8: Postman Request for User data retrieval (Valid data).	21
9: Postman Request for User data retrieval (invalid data).	22
10: Signup page after entering existing email.	23
11: Postman request for signing up existing user.....	23
12: Postman request for signing up new user.....	24
13: database snapshot after signing up new user.....	24
14: participants list after joining meeting with 3 users.	25
15: Meeting details in DB	26
16: Transcription file details in DB for user Tayma.....	26
17: Transcription file details in DB for user Rami	27
18: Two meetings with different ID's	28
19: Recording file.	29

1. Abstract

This project focuses on the development of a **Web Video Chat Application with Transcription**, designed to enhance communication by combining real-time video conferencing with automated transcription features. The application leverages **Stream.io SDK** for building reliable and scalable video calls and chat functionalities, ensuring a seamless user experience. Additionally, it integrates **AssemblyAI** for speech-to-text transcription, enabling users to review conversations after their calls. The project aims to create an efficient and user-friendly platform, addressing modern communication needs by merging video technology with AI-driven transcription. This document outlines the entire development process, including system architecture, API integration, and user interface design, while highlighting key challenges and solutions encountered during the implementation.

2.Introduction

In today's rapidly evolving digital landscape, the demand for seamless communication tools has never been greater. Video conferencing has become an integral part of both professional and personal life, facilitating real-time interactions regardless of physical location. This project, **Web Video Chat Application with Transcription**, aims to provide an innovative solution that not only allows users to connect via video calls but also offers a powerful feature: automatic transcription, enabling users to review their conversations afterward.

This application leverages **Stream.io SDK**, a leading solution for building scalable video chat and messaging systems. It enables high-quality, real-time video communication and chat functionalities, providing users with a smooth and engaging experience. To further enhance the utility of the platform, we integrated **AssemblyAI**, a state-of-the-art speech-to-text model that transcribes conversations in real-time. These transcripts are then saved for later review, making it easier for users to access and analyze their meetings and discussions.

Through this project, the goal was to create an efficient, user-friendly platform that addresses modern communication needs by combining advanced video technology with machine learning-driven transcription. This book details the process of designing, developing, and implementing the application, covering everything from backend architecture and API integration to the front-end user experience.

The following sections provide a comprehensive guide to the project, including an exploration of the technologies used, challenges faced, solutions implemented, and lessons learned throughout the development process. Whether you're a developer looking for technical insights or a user wanting to understand the application's capabilities, this book will serve as a detailed resource for understanding and utilizing this innovative video chat application.

3. Solution Description.

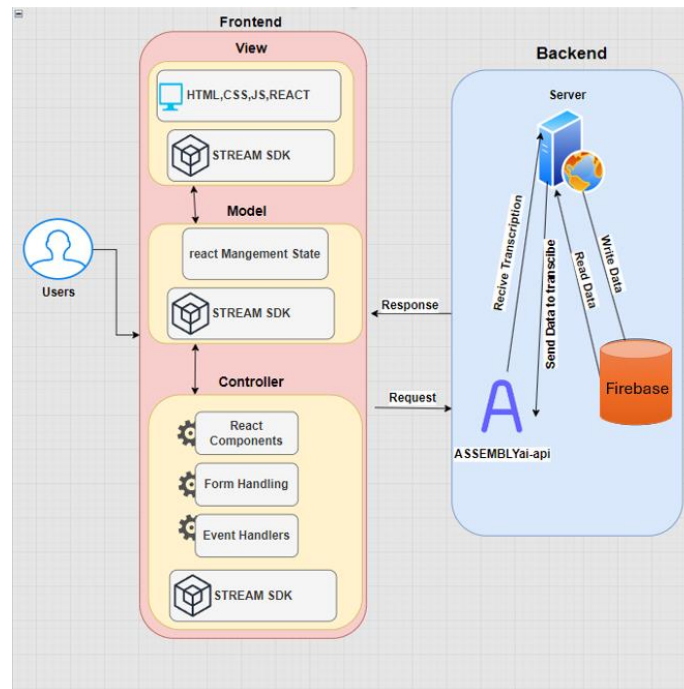
3.1. General Description.

The system was designed to facilitate seamless video communication combined with real-time transcription services. The primary goal was to provide users with the ability to participate in both one-on-one and group video calls while ensuring that conversations are transcribed accurately for future reference. To achieve this, we integrated the AssemblyAI API for speech-to-text conversion and the Stream.io SDK for video chat functionalities.

The system architecture supports both individual and group calls, with features such as microphone and camera control, encryption for user privacy, and storage of transcription files by date. This structure ensures that the system is scalable, robust, and efficient.

The target audience includes businesses, educators, and individuals seeking a reliable platform for video meetings with transcription capabilities, enhancing accessibility and record-keeping for future review.

3.2. Architecture Overview.



1: System Architecture

Our system architecture comprises several critical components that collectively enable the application's functionality:

1. Frontend:

- **Description:** The frontend represents the user interface of the application. It is built using React and provides the interactive elements and visual components through which users interact with the system.
- **Responsibilities:**
 - **User Interface (UI):** Presents data to the user and captures user inputs.
 - **Client-Side Logic:** Handles user interactions and communicates with the backend server via API calls.

2. Backend Server:

- **Description:** The backend server, implemented with Express.js, is responsible for processing client requests, executing business logic, and interfacing with external services and APIs.
- **Responsibilities:**

- **Request Management:** Processes incoming requests from the frontend and sends appropriate responses.
- **Service Integration:** Connects to external services, including video and chat APIs, and manages data exchanges with these services.
- **Authentication:** Oversees user authentication and authorization procedures.

3. Database:

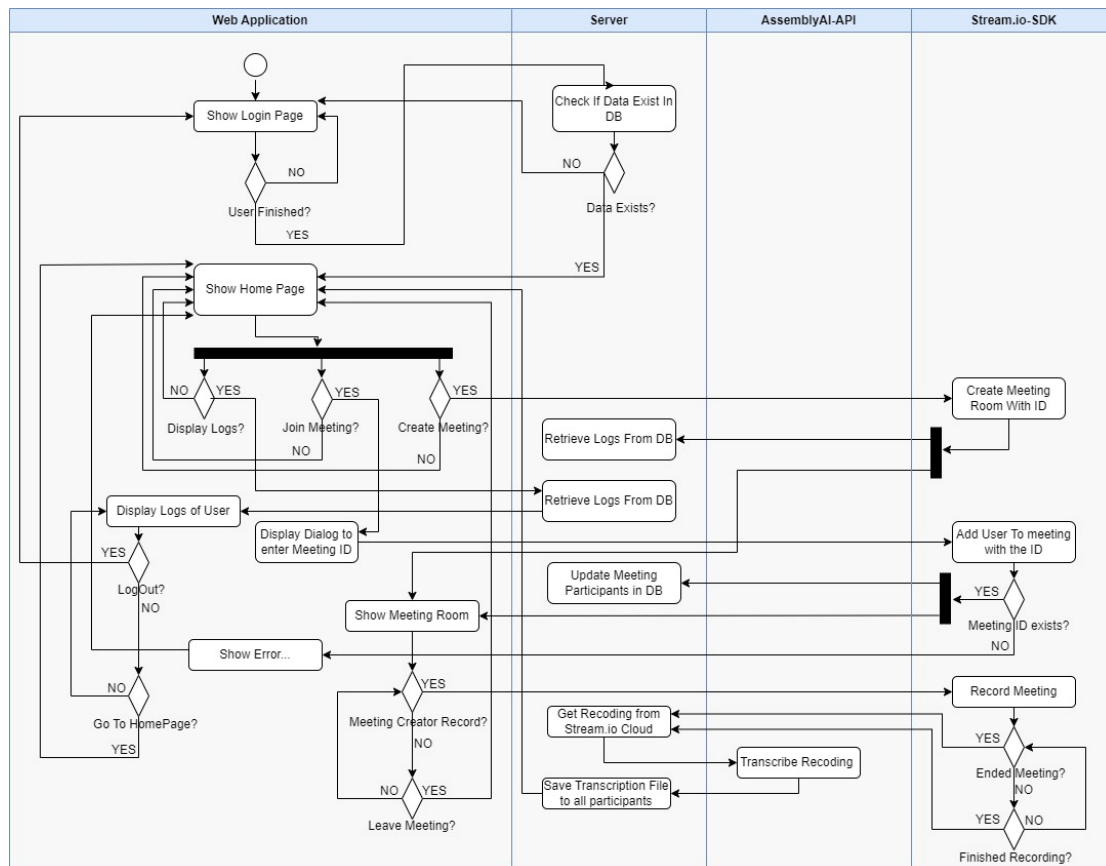
- **Description:** Firebase Firestore serves as the database, providing scalable, real-time data storage and retrieval.
- **Responsibilities:**
 - **Data Management:** Stores and manages user data, meeting details, and system logs.
 - **Real-Time Updates:** Delivers real-time updates to the frontend, ensuring current information is displayed.

4. APIs and SDKs:

- **Description:** A variety of APIs and SDKs are utilized to enhance functionality and streamline development processes.
- **Responsibilities:**
 - **Stream Video SDK:** Facilitates video call management and integration.
 - **Firebase APIs:** Handles tasks related to user authentication, real-time database operations, and other backend functions.
 - **AssemblyAI:** Provides transcription services for audio recordings.

3.2. Activity Diagram.

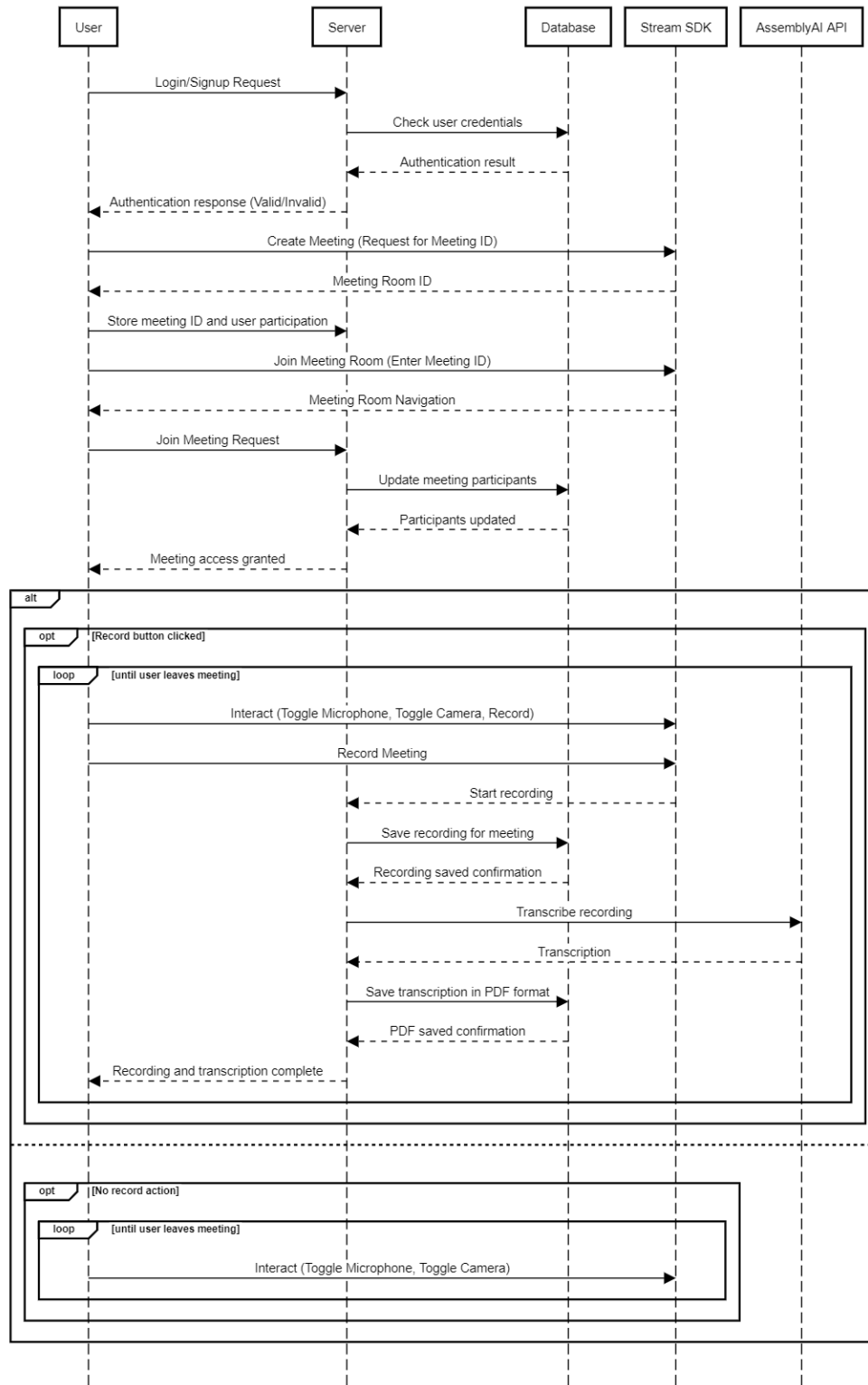
The following activity diagram shows the whole process from when the user opens the web application and until he finishes doing all what he wants in the application.



2: Activity Diagram

3.3. Sequence Diagram.

The following sequence diagram shows the logical flow between the user, server, web components and external API's and SDK's.



3: Sequence Diagram

4. Development Process.

4.1. What We Did to Develop and Build the System.

To develop our video chatting website with transcription capabilities, we undertook several key steps:

- **Requirements Gathering:** We started by defining the project requirements, focusing on integrating real-time video communication with high-quality speech-to-text functionality. We assessed user needs, including accessibility features, and mapped out the expected user experience.
- **Technology Selection:** We chose to utilize the AssemblyAI's model for its advanced speech recognition capabilities. This model was selected for its superior accuracy and ability to handle diverse acoustic environments and accents.
- **System Architecture:** We designed a robust system architecture to support video communication and transcription services. The architecture included a frontend developed with React for the user interface, a backend built with Express.js to manage API requests and handle server-side logic, and AssemblyAI's API for speech recognition.
- **Integration:** We integrated the AssemblyAI API into our backend to process audio from video calls and generate transcriptions. We also implemented functionality to send these transcriptions to participants after each meeting.
- **Testing and Refinement:** We conducted extensive testing to ensure the accuracy and performance of both the video communication and transcription features.

4.2. Tools Used.

- **Frontend:** React for building the user interface.
- **Backend:** Express.js for handling server-side logic and API management.
- **Speech Recognition:** AssemblyAI's model for transcription services, which supports English and meets our system's accuracy requirements.
- **Video Chatting:** Stream Video SDK for implementing video communication features.
- **Database:** Firebase Firestore for managing user data and meeting records.
- **Deployment:** Vercel for deploying both serverless functions and the client side. Vercel is chosen for its cost-effectiveness and suitability for our project's architecture.

4.3. Client interaction during development.

- **Regular Updates:** We maintained regular communication with clients through weekly progress meetings. These updates included demonstrations of completed features and discussions on any required adjustments.
- **Feedback Integration:** We actively sought feedback from clients and end-users throughout the development process. This feedback was used to make iterative improvements to the system and ensure it met user needs and expectations.
- **Testing and Validation:** We involved clients in the testing phase, providing them with access to test versions of the application to gather real-world feedback and make necessary refinements.

4.4. Challenges Encountered and How We Addressed Them.

- **Accuracy of Transcriptions:** One significant challenge was ensuring high accuracy in transcriptions across different accents and noisy environments. To address this, we utilized the AssemblyAI model, which is known for its advanced capabilities and robustness. Although we did not fine-tune the model, we relied on its inherent strengths to handle diverse accents and varying acoustic conditions effectively. By leveraging the advanced features of the AssemblyAI model, we aimed to achieve reliable and precise transcriptions in real-world scenarios.
- **Scalability:** Ensuring the system could handle many concurrent users posed a challenge. We used the Stream.io SDK for implementing the video chatting functionality, which effectively manages scalability and ensures reliable performance during peak usage. The SDK's robust infrastructure allows us to handle high volumes of simultaneous video calls without compromising on quality.
- **User Privacy:** Handling sensitive user data required stringent privacy measures. We implemented strong encryption protocols for data transmission and storage and complied with relevant data protection regulations. Specifically, we encrypt user passwords using bcrypt to ensure they are securely stored in our database.

4.5. Decisions Made.

- **Transcriptions language:** we decided to focus our project on English-speaking users because the AssemblyAI model doesn't support Hebrew language.
- **User Experience:** We chose to send transcriptions directly to users post-meeting to enhance accessibility and user engagement. This decision was based on user feedback and our goal to improve the post-chat experience.
- **Gateway Timeout Configuration:** We decided to set the gateway timeout in Vercel to 60 seconds to accommodate long-running tasks such as transcribing the meeting audio and generating the PDF file with the transcription results. This ensures that these time-intensive operations have sufficient time to complete without being prematurely terminated by the server.

5. Verification and Evaluation.

- We will evaluate our web application based on its ability to capture high quality recordings and sending them appropriately for AssemblyAI API, and writing the transcription as an organized pdf file and saving the generated file for each participant.
- Due to the nature of our project architecture and development process, we will perform testing on 4 modules to ensure right functionalities and good user serving. The modules are Web application, server AssemblyAI API and stream.io SDK.

Test	Module	Tested Function	Expected Result
1	Web application	Page load	First page load<2s
2	Web application	UI	Responsive and easy to use.
3	Web application	Logs page	Correctly show files and enable user to perform filtering to results.
4	Web application	display meeting	Correctly display the meeting room.
5	Web application	Join meeting	Display pop out asking for meeting ID.
6	Web application	Log out	Correctly navigate to login page.
7	Web application	Sign in	Correctly navigate to homepage.
8	Server	Login	Retrieve correct result from DB.
9	Server	Sign-Up	Correctly insert new user to DB.
10	Server	Navigation	Correct navigation between routes.
11	Server	Update participants	Correctly update participant for specific meeting.
12	Server	Transcription file saving	Correctly save and structure transcription files in DB for each participant.

13	Stream.io SDK	Creating meeting	Correctly create meeting room connection with specific ID.
14	Stream.io SDK	Meeting recording	Correctly record the meeting audio.

1: Table of Tests.

5.1. Testing Process

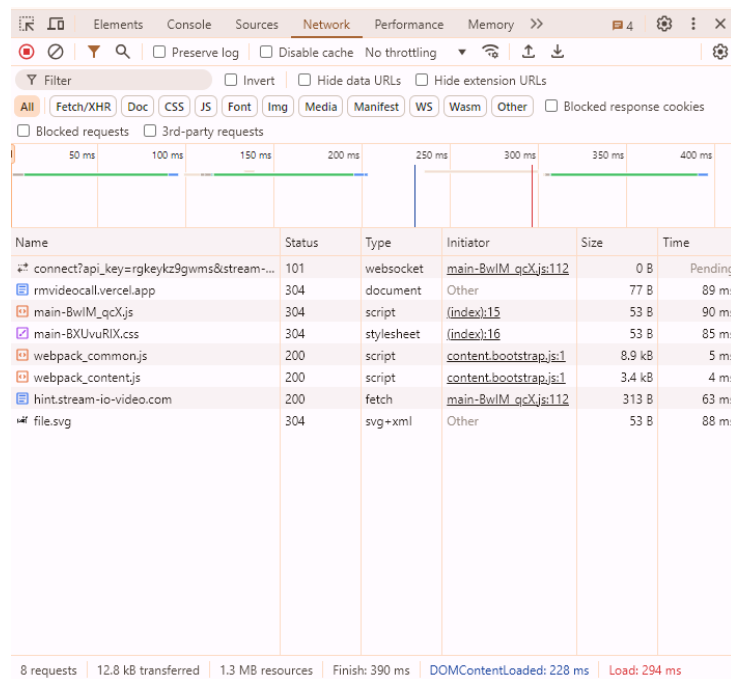
In the testing phase of our project, we conducted a series of manual tests to ensure the functionality and performance of the web application, server, and integrated APIs. These tests included evaluating page load times, user interface responsiveness, correct display and navigation between pages, and the accuracy of operations such as logging in, signing up, updating participants, and saving transcription files. Additionally, we tested the integration with external services like AssemblyAI's API for transcription accuracy and Stream.io SDK for creating, recording, and sending meeting recordings.

Manual testing allowed us to interact with the application in real-time, providing immediate feedback on user experience and functionality. One of the main advantages of manual testing is its flexibility, as it allows testers to quickly adapt to any changes in the system and identify issues that automated tests might overlook, especially in complex scenarios like user interface interactions. This approach helped us ensure that the application meets our expectations in terms of usability, performance, and integration with third-party services.

5.2. Testing Results

5.2.1 Page Load

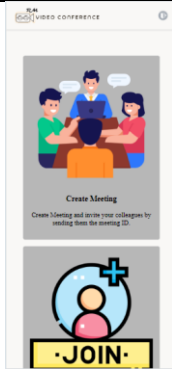
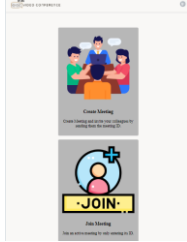

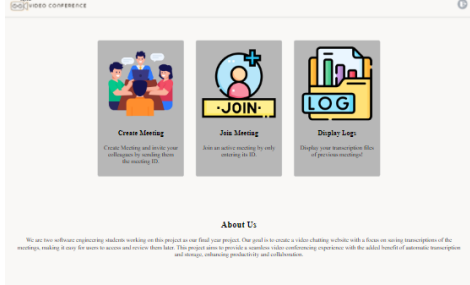
- **Tested Function:** First page load time.
- **Expected Result:** The first page should load in less than 2 seconds.
- **Test method:** google dev tools, take the page load time from performance tab.
- **Result:** Passed, with a loading time of approximately 400 ms, indicating quick responsiveness.



4: Page Load Test Result

5.2.2 UI

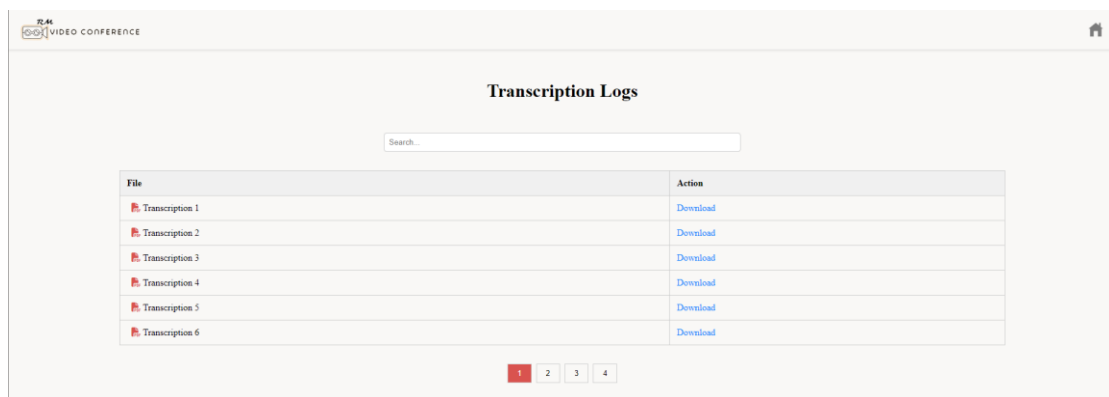
- **Tested Function:** User Interface responsiveness and ease of use.
- **Expected Result:** The UI should be responsive and user-friendly.
- **Test method:** google dev tools using the devices options.
- **Result:** Passed, showing that the application provides a smooth and intuitive user experience.

Device type	Snapshot
Iphone XR	
ipad mini	
Asus zenbook fold	
Nest hub max	

2: Table of screenshots for the website using different devices

5.2.3 Logs Page

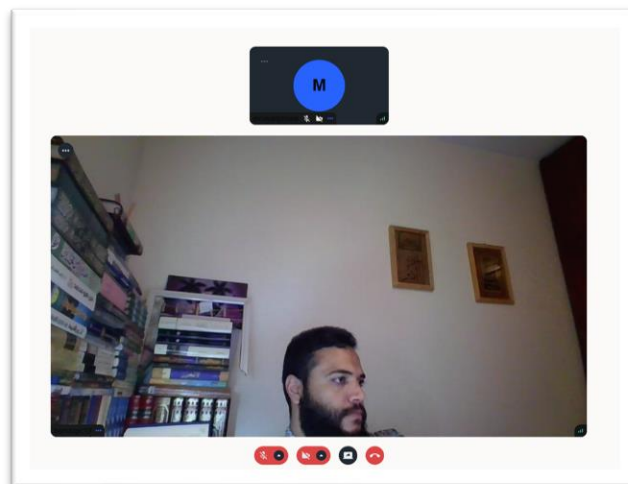
- **Tested Function:** Display of transcription files and filtering options.
- **Expected Result:** The page should correctly show files and enable users to perform filtering on the results.
- **Test method:** manually open the logs page and try to filter.
- **Result:** Passed, confirming that users can effectively manage and filter transcription logs.



5: Logs Screen

5.2.4 Display Meeting

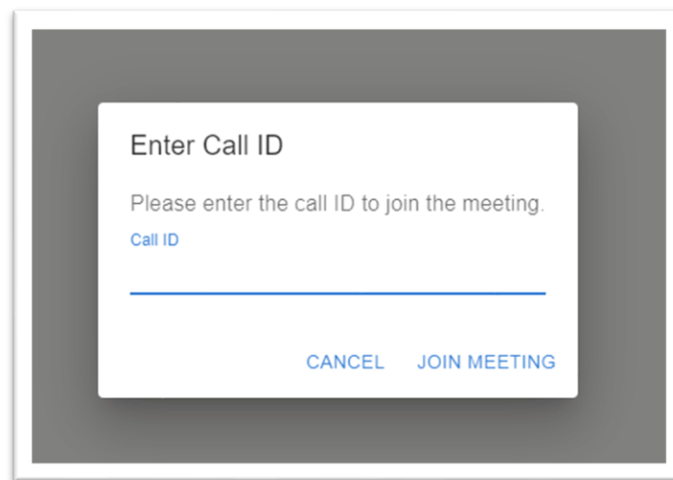
- **Tested Function:** Meeting room display.
- **Expected Result:** The meeting room should be correctly displayed to users.
- **Test method:** click on create meeting and check if the meeting room displayed well.
- **Result:** Passed.



6: Meeting Room

5.2.5 Join Meeting

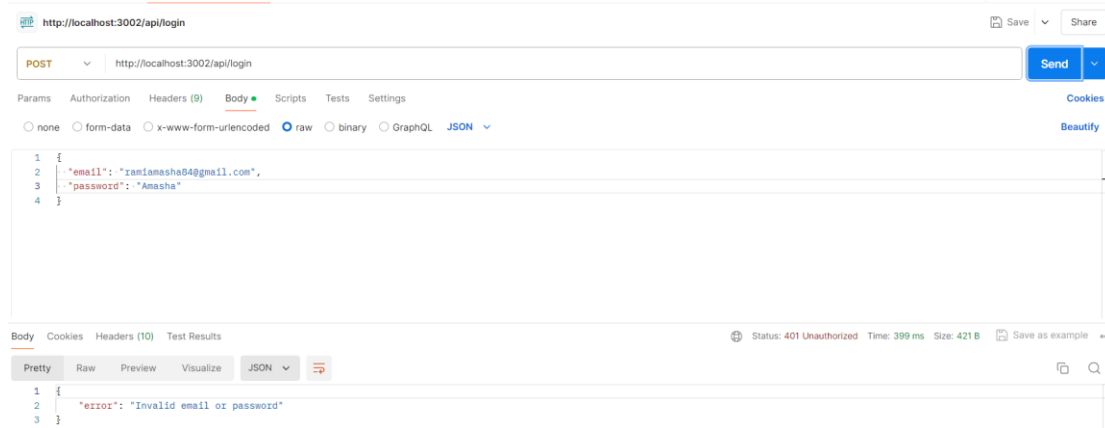
- **Tested Function:** Join meeting process.
- **Expected Result:** A pop-up should appear asking for the meeting ID.
- **Test method:** click on join meeting and check if a pop out displayed well and asks for meeting ID.
- **Result:** Passed, indicating that the user is correctly prompted to input a meeting ID when joining a meeting.



7: Join Meeting Dialog

5.2.6 Log Out

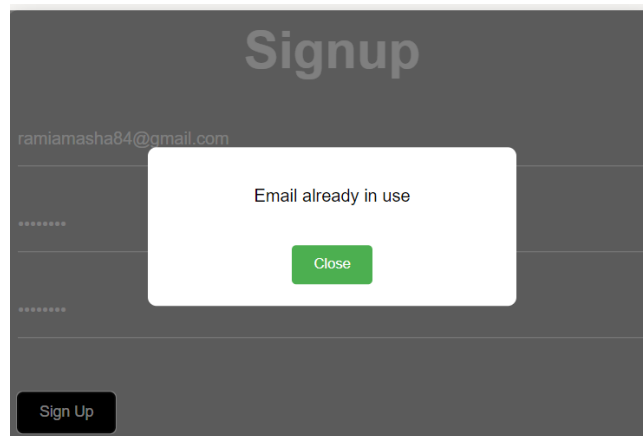
- **Tested Function:** User logout functionality.
- **Expected Result:** Correct navigation to the login page after logging out.
- **Test method:** click on the logout icon and check if the user being navigated to login page.
- **Result:** Passed, confirming that users are redirected to the login page upon logging out.



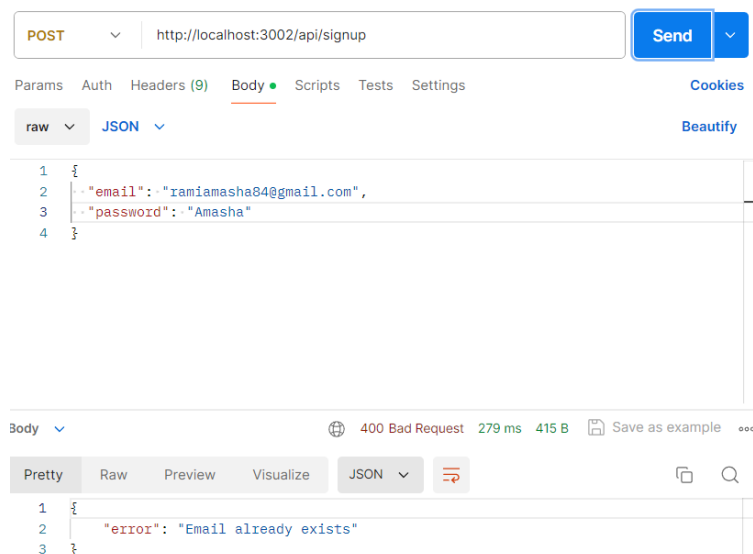
9: Postman Request for User data retrieval (invalid data).

5.2.9 Sign-Up

- **Tested Function:** Insertion of new user data into the database.
- **Expected Result:** Successfully add a new user, with a warning displayed if the email is already in use.
- **Test method:** manually entering an existing email and try to sign up and to send request with postman for registering new email and check the response.
- **Result:** passed.
 - **Existing email:**

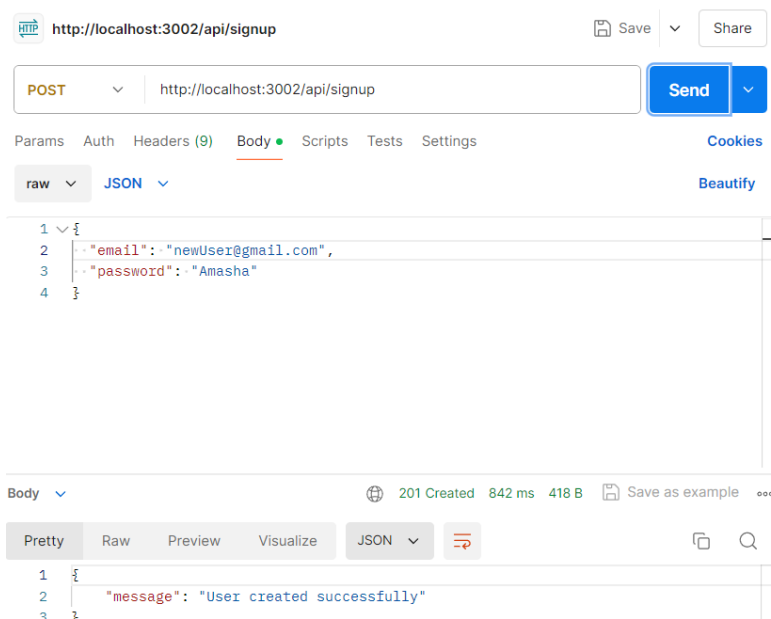


10: Signup page after entering existing email.

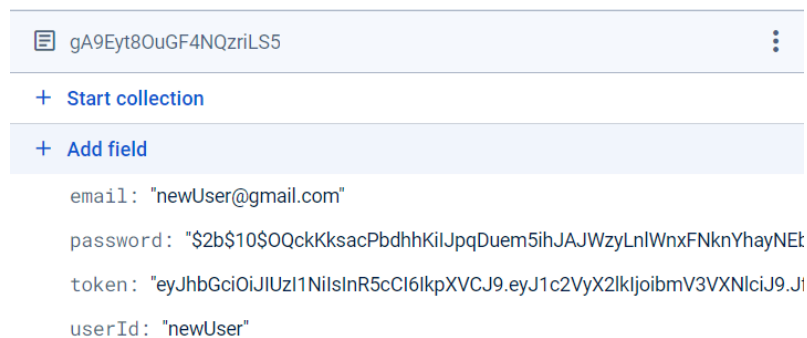


11: Postman request for signing up existing user

- Register new user:



12: Postman request for signing up new user



13: database snapshot after signing up new user

5.2.10 Navigation

- **Tested Function:** Server-side route navigation.
- **Expected Result:** Correct navigation between routes.
- **Result:** Passed, confirming seamless server-side routing.

5.2.11 Update Participants

- **Tested Function:** Updating participants for a specific meeting.
- **Expected Result:** Correctly update participant information for the specified meeting.
- **Test method:** manually enter meeting with multiple users and check if the list in the data base is updated correctly.
- **Result:** Passed, indicating accurate participant updates.

Entered meeting with ID 58wtnlajy with 3 different users: taim, ramiamasha84 and tayma.

```
callId: "call-58wtnlajy"
createdAt: "2024-09-09T18:53:01.568Z"
participants
  0 "taim"
  1 "ramiamasha84"
  2 "tayma"
type: "default"
```

14: participants list after joining meeting with 3 users.

5.2.12 Transcription File Saving

- **Tested Function:** Saving and structuring transcription files in the database for each participant.
- **Expected Result:** Correctly save and organize transcription files for each participant.
- **Test method:** enter meeting with more than one user, record the meeting and check if all users got the transcription file correctly.
- **Result:** passed.

Meeting information:

```
callId: "call-2tussomv7"
createdAt: "2024-09-10T17:44:27.879Z"
participants
  0 "ramiamasha84"
  1 "taymaibrahem"
type: "default"
```

15: Meeting details in DB

```
+ Add field
email: "taymaibrahem@gmail.com"
logs
  0 "https://firebasestorage.googleapis.com/v0/b/finalproje (string) 56ffd.appspot.com/o/transcriptions%2Ftranscription_call-2tussomv7.pdf?alt=media&token=83b3b4c5-ea79-4658-a218dec991"
password: "$2b$10$P$UtzccctoMZDXEh.boRtev4ammSPTTUdof/.OW6k4Xo4Ad1Hc
token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkjoidGF5bWFPYnJha
-y4e6vAoOPTOePN12ZOzVdsA8l11bSSUCE"
userId: "taymaibrahem"
```

16: Transcription file details in DB for user Tayma

+ Add field



email: "ramiamasha84@gmail.com"

(string)



▼ logs

0 "https://firebasestorage.googleapis.com/v0/b/finalproject-56ffd.appspot.com/o/transcriptions%2Ftranscription_call-eh2ml9euf.pdf?alt=media&token=9779dbb6-79ba-4e29-a2ad-51d2bb77849a"

1 "https://firebasestorage.googleapis.com/v0/b/finalproject-56ffd.appspot.com/o/transcriptions%2Ftranscription_call-rpic11e0p.pdf?alt=media&token=0433f94c-4b82-4b41-bf933c208f95" (string)  

2 "https://firebasestorage.googleapis.com/v0/b/finalproject-56ffd.appspot.com/o/transcriptions%2Ftranscription_call-2tussomv7.pdf?alt=media&token=83b3b4c5-ea79-4658-b095-a218decbc991"

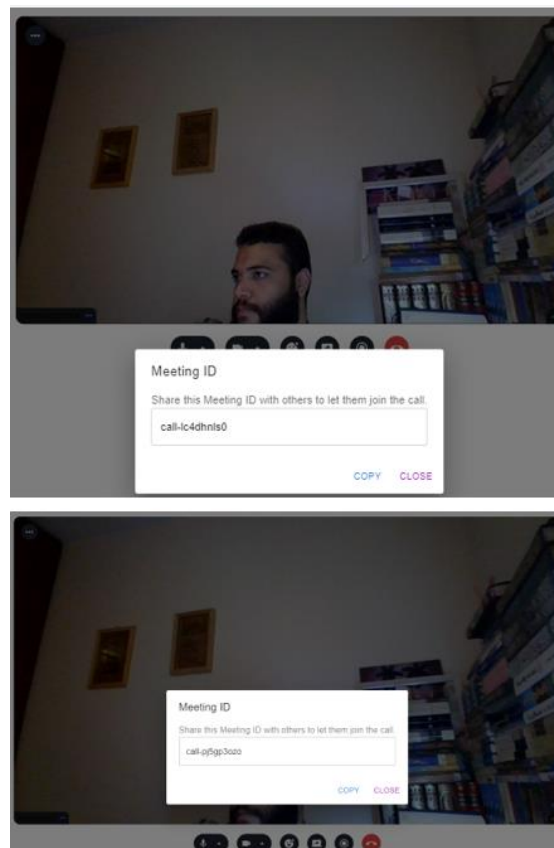
3 "https://firebasestorage.googleapis.com/v0/b/finalproject-56ffd.appspot.com/o/transcriptions%2Ftranscription_call-

17: Transcription file details in DB for user Rami

5.2.13 Creating Meeting

- **Tested Function:** Creation of a meeting room with a specific ID.
- **Expected Result:** Correctly create a meeting room connection with the given ID.
- **Test method:** create two meetings at the same time and verify they have different ID's.
- **Result:** Passed, confirming successful meeting room creation.

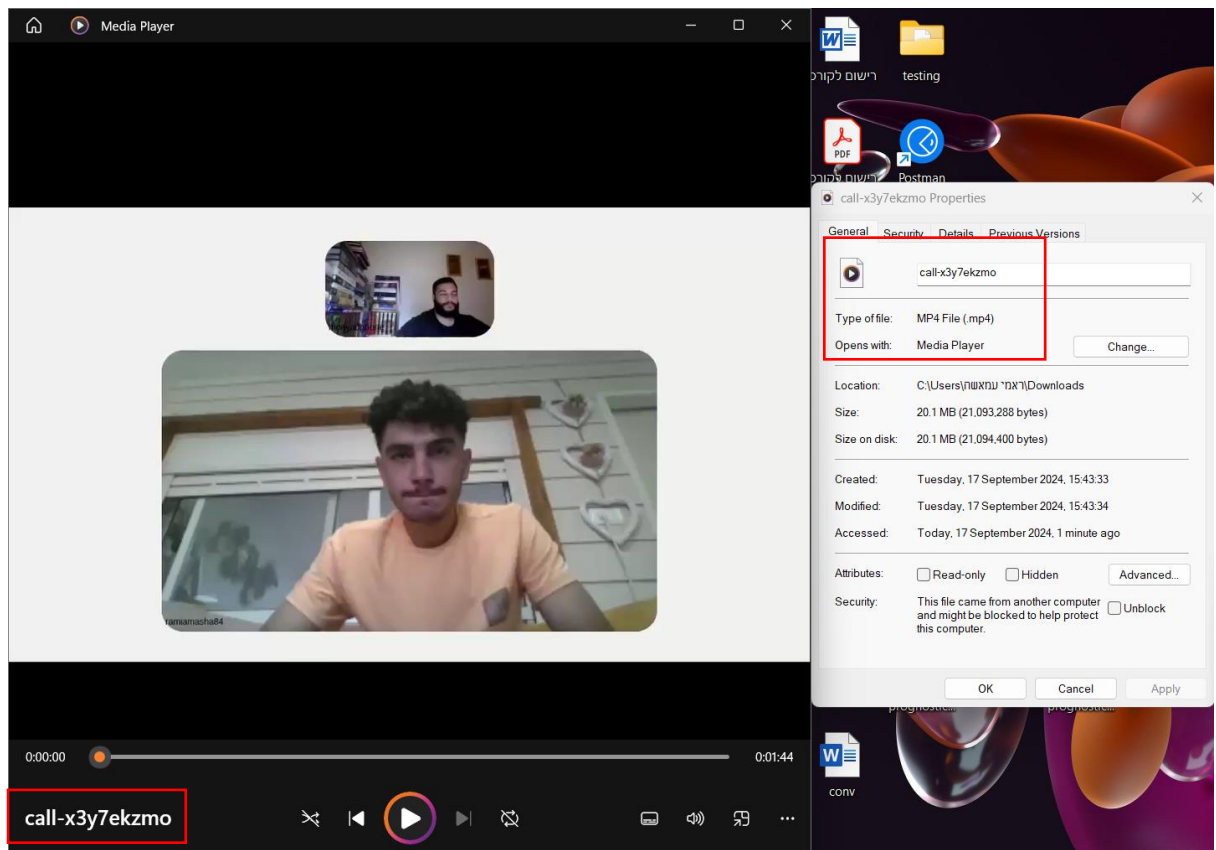
Two meetings at the same time, have different meeting IDs.



18: Two meetings with different ID's

5.2.14 Meeting Recording

- **Tested Function:** Recording of the meeting audio.
- **Expected Result:** Correctly record the meeting audio.
- **Test method:** manually record meeting and check if the meeting has been recorded successfully.
- **Result:** passed.



19: Recording file.

6. Outcomes and Conclusions.

6.1. Project Goals: Explanation and Justification.

We successfully achieved the project goals by meeting both the functional and non-functional requirements.

1. Functional Requirements:

- Implemented **one-on-one and group video call functionalities** using the Stream SDK, providing smooth and reliable video communication.
- Integrated **AssemblyAI's API** for accurate real-time transcription, ensuring high-quality speech-to-text conversion.
- Provided user controls for **microphone and camera toggling**, allowing users to manage their video/audio input seamlessly.
- Ensured **user privacy** by encrypting sensitive data, including passwords, using bcrypt.
- Organized transcription files by their creation date for **easy access** and management post-meetings.

2. Non-Functional Requirements:

- Maintained **low latency** and high responsiveness, ensuring a smooth user experience during video calls without interruptions.
- Designed the system architecture to **scale efficiently**, accommodating increasing numbers of users and video sessions.
- Ensured the application is **robust and stable** through comprehensive error handling, preventing crashes and ensuring continuous availability.
- Achieved **compatibility** across various devices and operating systems, offering a consistent experience for all users.
- Developed a well-structured, **documented codebase**, ensuring the system can be easily maintained and enhanced in the future.

These outcomes confirm that the project met its goals and adhered to the performance and quality standards set at the outset.

6.2. Addressing challenges.

We have addressed key challenges effectively (as detailed in sections [4.4](#) and [4.5](#)):

- To ensure **transcription accuracy**, we used AssemblyAI's Conformer-2 model to handle diverse accents and noisy environments.
- For **scalability**, the Stream SDK allowed us to manage large numbers of concurrent users efficiently.
- Implemented encryption protocols to protect user data and comply with regulations.
- Set a **60-second gateway timeout** in Vercel to support long-running tasks like transcription processing.

6.3. Decision making considerations.

- We chose AssemblyAI based on its high accuracy for transcribing speech to text and the availability of a free tier, making it a cost-effective solution for our project while maintaining excellent performance.
- We chose Stream SDK based on its simplicity of integration into our codebase, its built-in components that streamline development, and its scalability to handle growing user demands. The SDK also offers high-quality, secure video communication, which aligns with our project's requirements for reliability and performance. Additionally, the free tier provided by Stream makes it a cost-effective solution for our project's initial phase.
- User feedback influenced our decision to send **transcriptions post-meeting**, improving engagement.
- We accounted for cases involving long meetings and recordings, where a 10-second gateway timeout would be insufficient, so we extended the timeout to ensure uninterrupted completion of tasks like audio transcription and PDF generation.

Overall, we delivered a robust and scalable system that fulfilled both functional and non-functional requirements, while effectively managing challenges and making well-reasoned decisions throughout the development process

7. Lessons learned.

The project was executed successfully. However, reflecting on the process reveals areas for potential enhancement:

- **User Interface (UI):** Although the application is functional and stable, the user interface could be more intuitive and visually appealing. Enhancing UI design could significantly improve user experience and engagement.
- **Language Support:** Expanding language support to include other languages available through AssemblyAI could broaden the application's accessibility and usability.
- **Broader Feedback:** We gathered valuable feedback from friends, family, and relatives. However, expanding the feedback pool to a larger, more diverse community would help ensure the application meets the needs of a broader audience. Conducting surveys with this wider group could inform further improvements.

7.1. Success criteria.

We successfully met the project criteria we defined at the outset. The speech-to-text transcription system achieved a high level of accuracy, capturing various speech nuances as outlined in the accuracy criterion. Additionally, the user experience was validated through usability tests and user feedback, with users reporting that the interface was user-friendly and easy to navigate. The transcription process after each video call was automated efficiently, delivering transcriptions within a reasonable timeframe. Furthermore, the integration with AssemblyAI was smooth, with no significant disruptions, and the performance remained consistent throughout the process.

8. Conclusions.

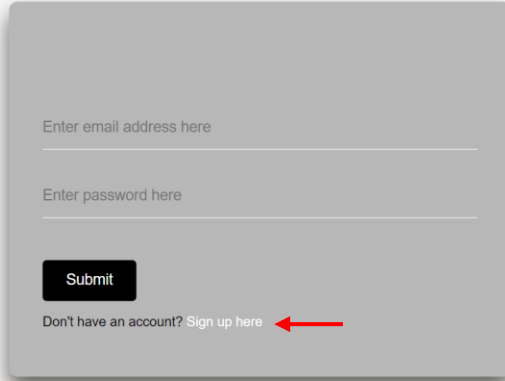
In conclusion, the project has been a success, achieving all the predefined goals and criteria. The integration of Stream SDK and AssemblyAI API resulted in a highly functional and scalable application that meets both the technical and user experience expectations. The system demonstrated high accuracy in transcription, efficient video communication, and reliable performance, aligning with the project's objectives.

Reflecting on the lessons learned, the main areas for future improvement include enhancing the UI design for better user engagement, expanding language support to cater to a more diverse audience, and seeking broader feedback to ensure the application meets the needs of a larger user base. These insights will guide future iterations and improvements, ensuring continued success and relevance of the application in a rapidly evolving technological landscape.

9. User Guide.

This section provides detailed operational instructions for using the system, focusing on successful use cases. It describes the typical user processes and interactions within the application to help users efficiently navigate and utilize the system's features.

1. Our application requires from you an account to use it, so at first you must create an account by Pressing on “Sign up here” button:



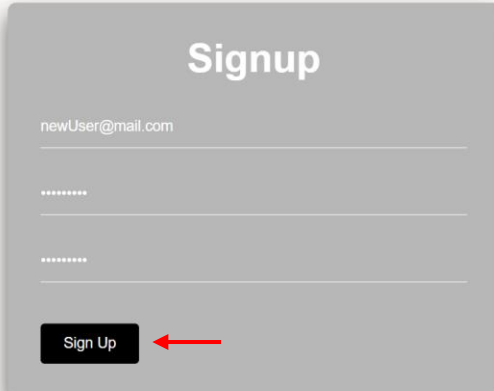
Enter email address here

Enter password here

Submit

Don't have an account? Sign up here

2. All you have to do is to enter your email and your password and then press on “Sign Up” button, this will create a private account for you so you can use later (Don't forget your password):

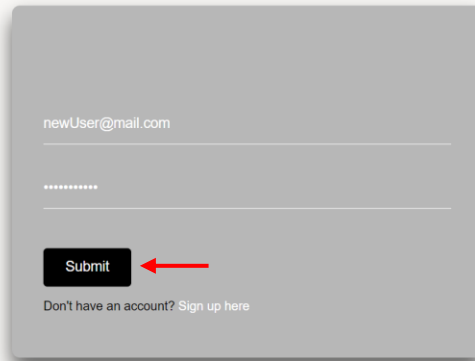


Signup

newUser@mail.com

Sign Up

3. After finishing signing up, you will be directed to the login page, now you can enter your email and password, and press on “Submit” to login:

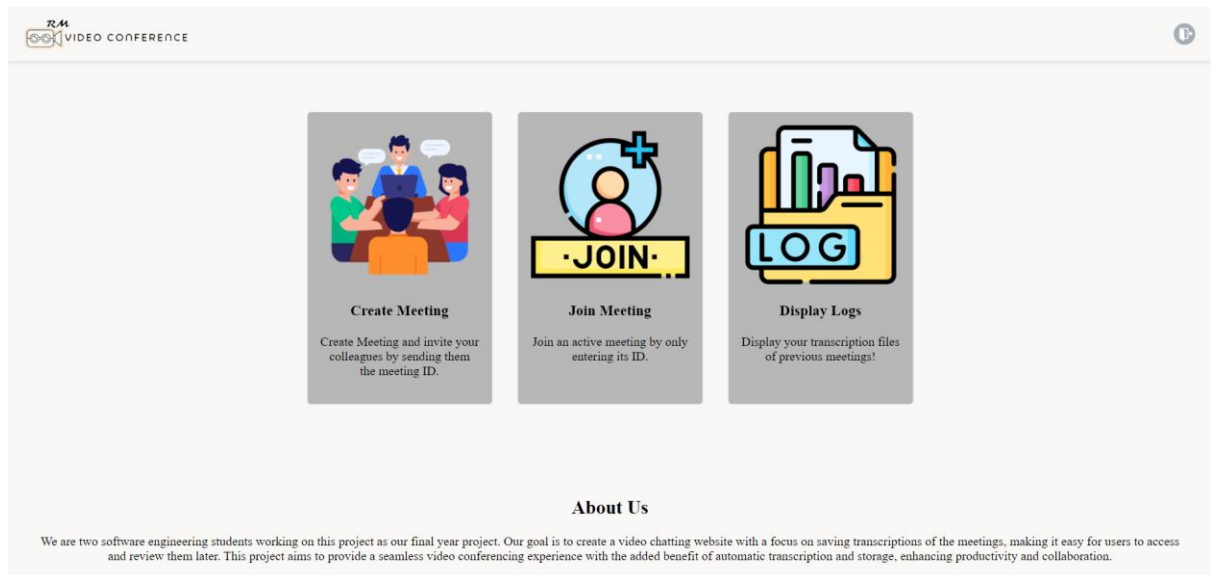



newUser@mail.com

Submit

Don't have an account? [Sign up here](#)


4. After logging in, you will be directed to your personal space. Here, you can create your own meeting, join another meeting, or check the transcriptions of your previous meetings:






Create Meeting

Create Meeting and invite your colleagues by sending them the meeting ID.



Join Meeting

Join an active meeting by only entering its ID.



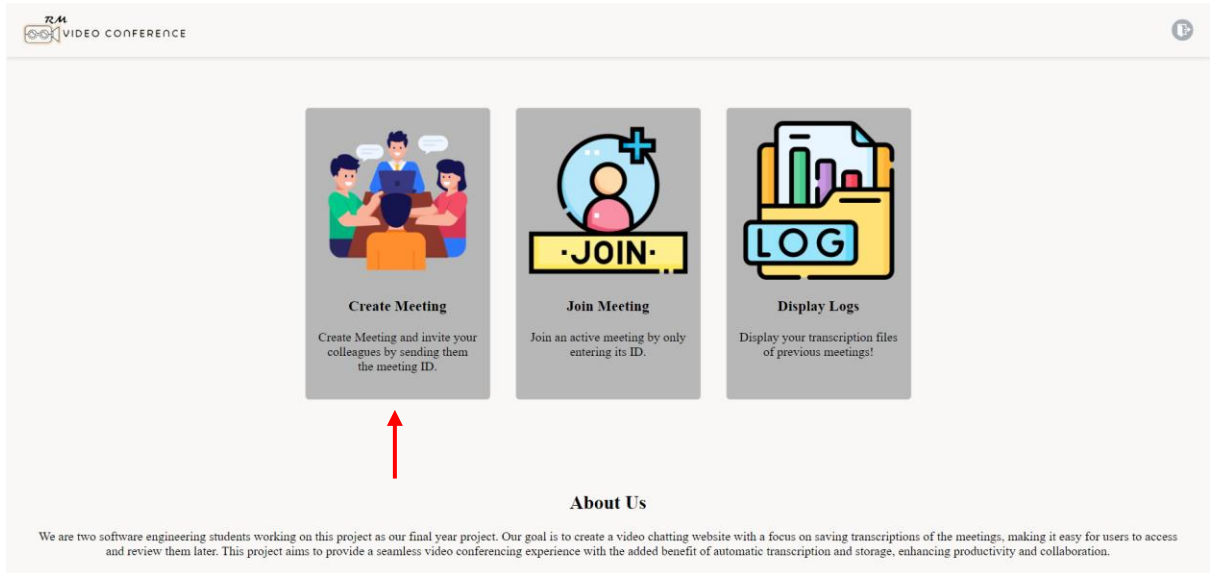
Display Logs

Display your transcription files of previous meetings!

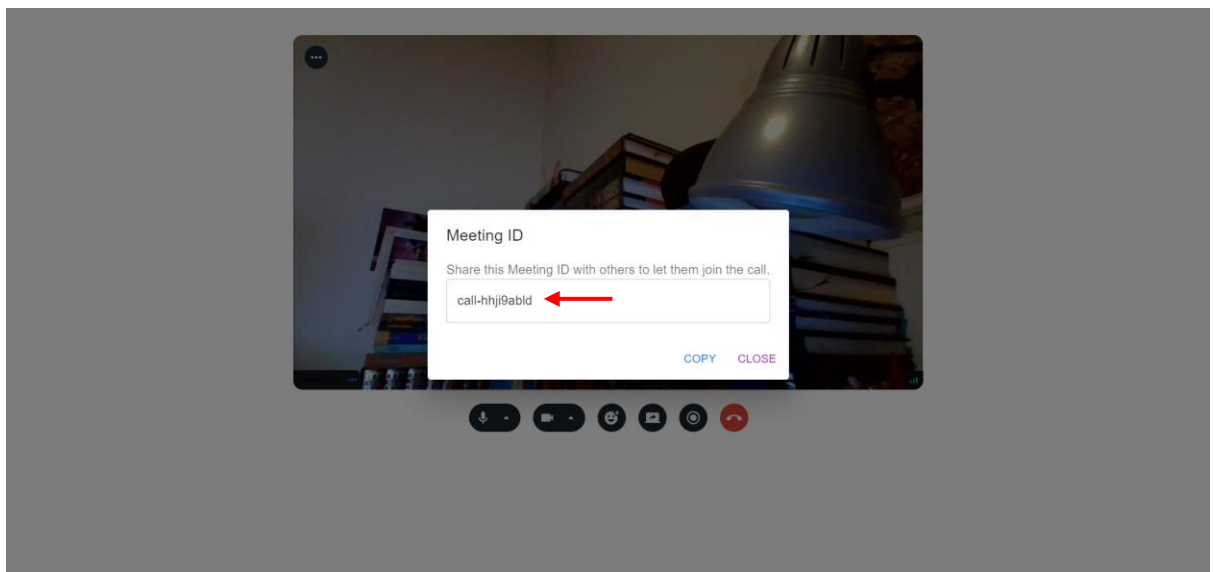
About Us

We are two software engineering students working on this project as our final year project. Our goal is to create a video chatting website with a focus on saving transcriptions of the meetings, making it easy for users to access and review them later. This project aims to provide a seamless video conferencing experience with the added benefit of automatic transcription and storage, enhancing productivity and collaboration.

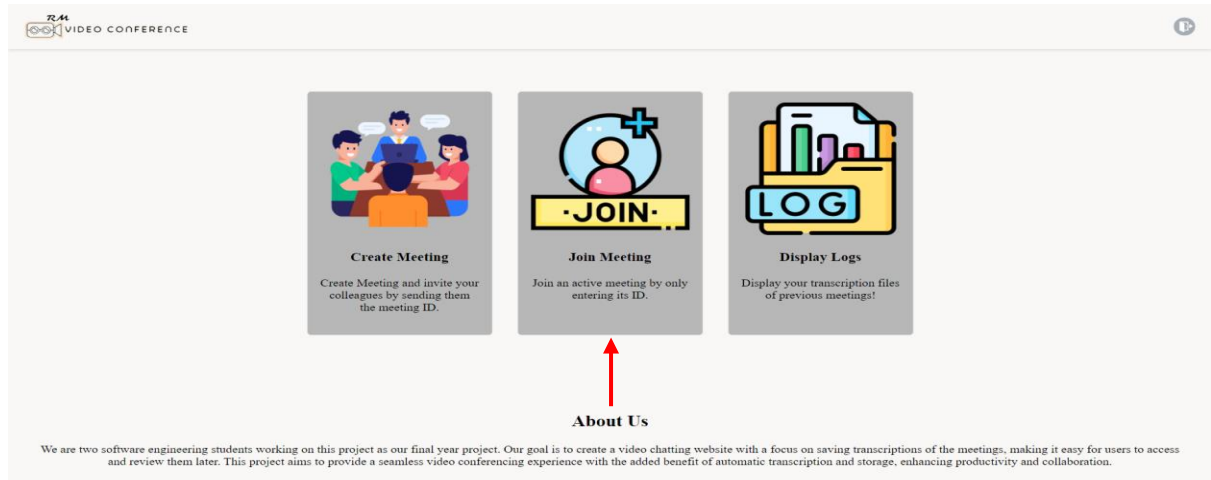
- 4.1. **Creating a meeting room:** you can create your own meeting room by pressing on the “Create Meeting” button, this will create a meeting room with its unique ID:



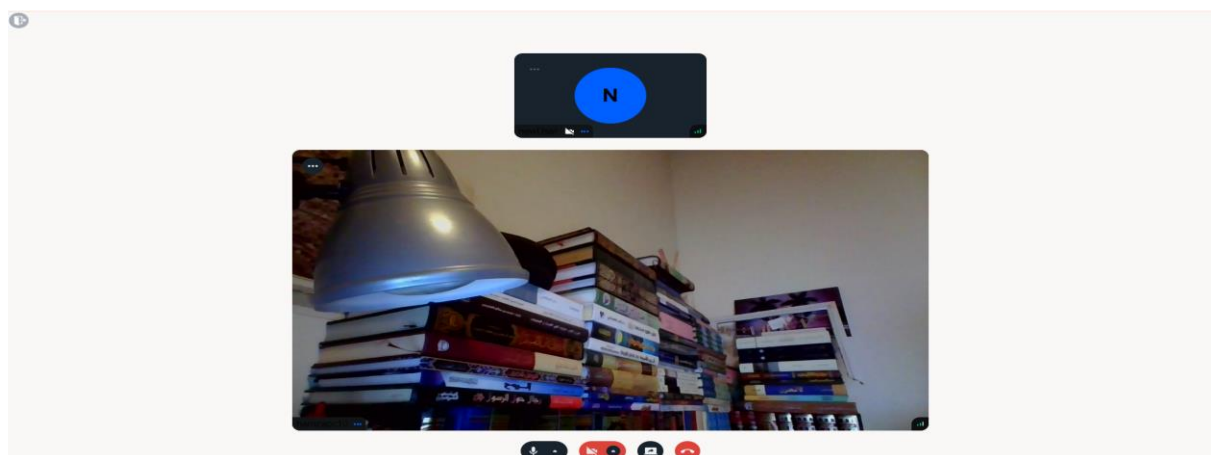
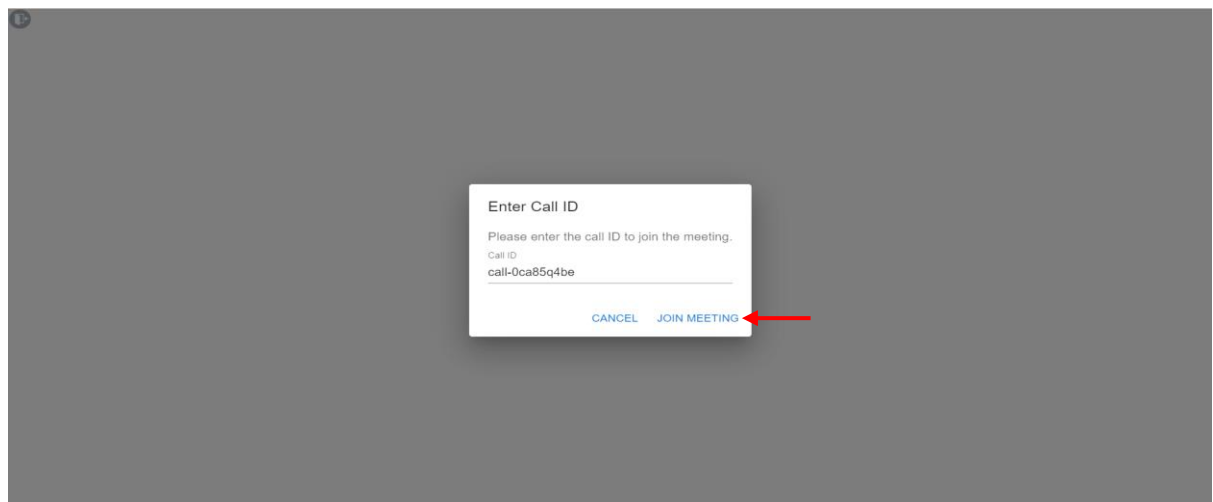
You can copy the meeting ID and send it to others so they can join you:



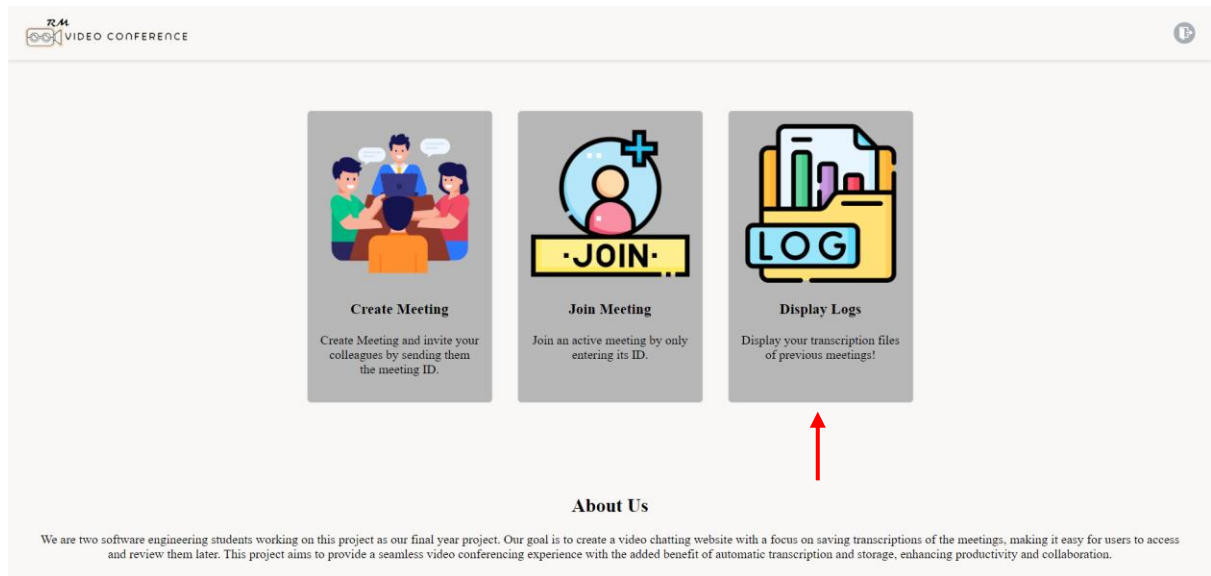
- 4.2. **Joining an active meeting room:** you can join another meeting by pressing on “Join Meeting” button (you must have the meeting ID):



This will pop up a window asking for the meeting ID, just enter the ID and press the “Join Meeting” button, and you will be directed to the meeting room:

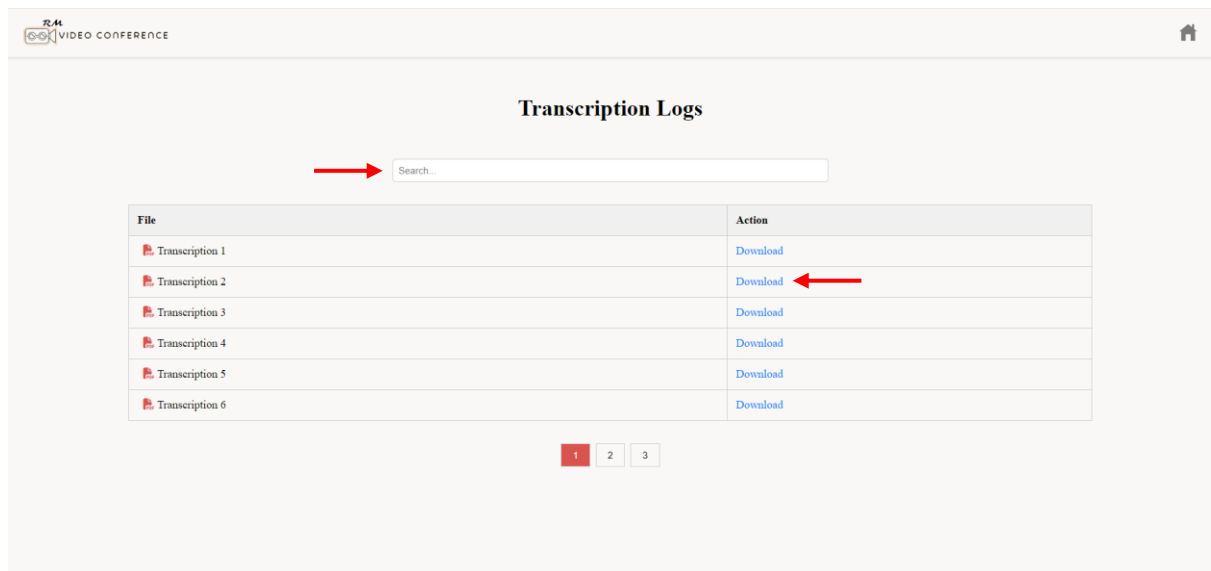


- 4.3. **Displaying the logs:** you can display your transcriptions of previous meetings by pressing on the “Display Logs” button:



You will be directed to the Transcription Logs page, where you can search for a specific Transcription and read it by clicking on the “Download” button.

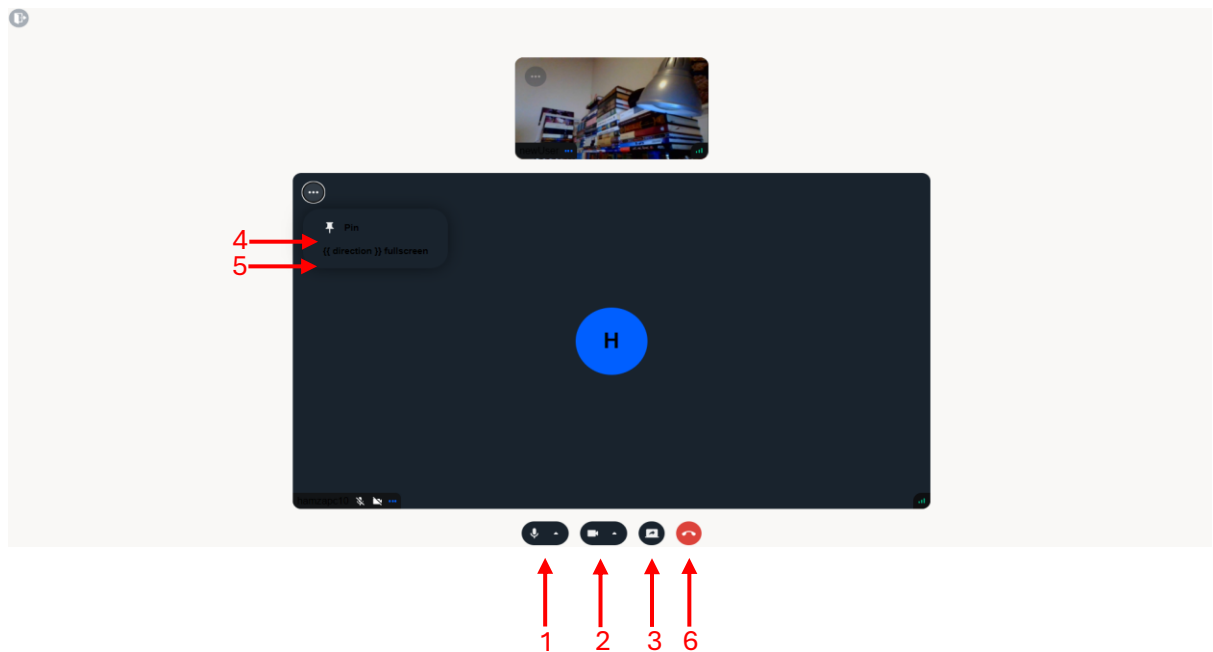
You can also search for a Transcription in the search bar:



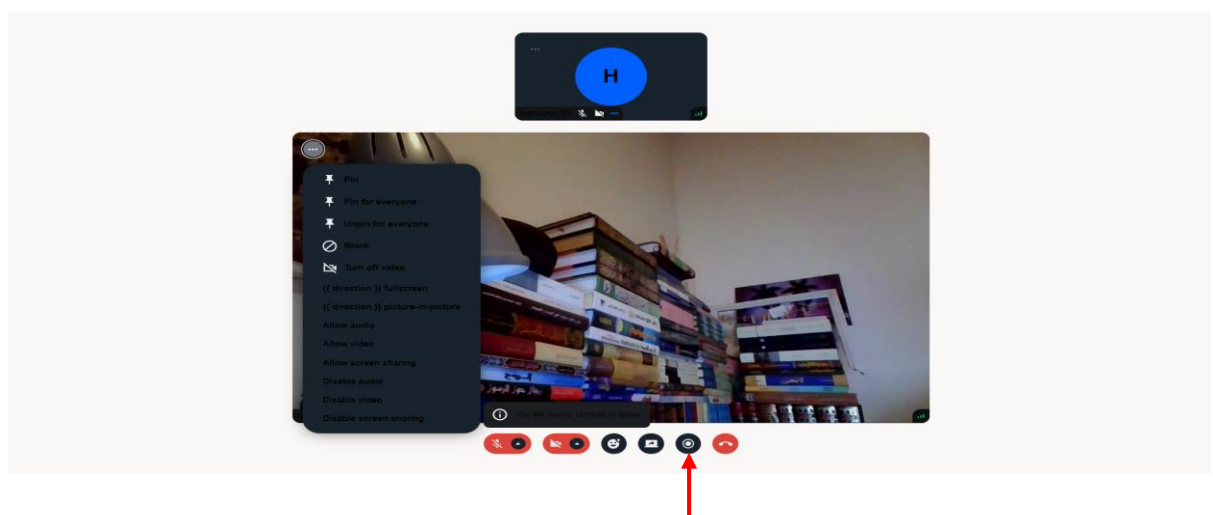
5. Inside a meeting room:

Inside the meeting room, the user can:

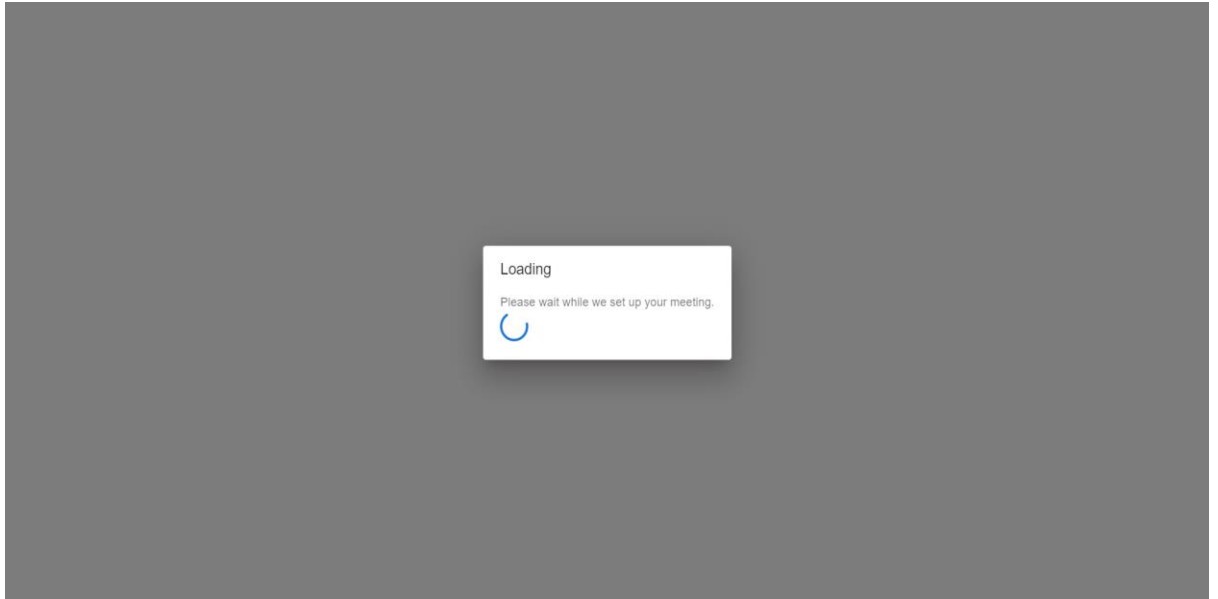
1. Turn On/Off his microphone.
2. Turn On/Off his camera.
3. Share his screen.
4. Pin a specific window.
5. Enter Fullscreen mode.
6. Exit the meeting.



In addition to that, the creator of the meeting, have more advantage, he's able to Allow/Disable the Camera/Microphone/Screen Sharing of other users, Block Users, Pin Users. And he is the one that can record the audio so it can **create a Transcription** at the end of the meeting by pressing on the button:



6. At the end of the meeting, the website will create the Transcription, and it will be added to the Transcription Logs (explanation at 4.3):



10. Maintenance guide.

10.1. Purpose.

The purpose of this maintenance guide is to ensure the continued usability and operation of the project after its completion. This guide provides instructions for applying updates, making changes, and improving the system to extend its lifecycle and usage.

10.2. System environment.

10.2.1. Operating system.

- compatible with windows, macOS and Linux.

10.2.2. Software requirements.

- Node.js: version v20.15.0. Download Link: <https://nodejs.org/en>
- Npm: for managing project dependencies. (Included with node.js)
- Vite + React for the frontend.
- Express.js for the backend API server.
- Firebase Firestore: used as cloud database.
- Git: for version control and code management.

10.2.3. Hardware requirements.

- Internet connection: required for firebase services and deploying changes.

10.3. Software installation and setup.

1. Clone the repository:

```
Git clone https://github.com/RamiAmasha31/rm-video-call  
Cd video-call
```

2. Install dependencies:

```
npm install
```

```
cd server  
npm install
```

3. Environment variables setup:

Create a .env file inside the server folder and provide the following values:

```
# Server Configuration  
SERVER_IP=localhost  
SERVER_PORT=3002  
  
# Firebase Configuration  
FIREBASE_API_KEY=YOUR_FIREBASE_API_KEY  
FIREBASE_AUTH_DOMAIN=YOUR_FIREBASE_AUTH_DOMAIN  
FIREBASE_PROJECT_ID=YOUR_FIREBASE_PROJECT_ID  
FIREBASE_STORAGE_BUCKET=YOUR_FIREBASE_STORAGE_BUCKET  
FIREBASE_MESSAGING_SENDER_ID=YOUR_FIREBASE_MESSAGING_SENDER_ID  
FIREBASE_APP_ID=YOUR_FIREBASE_APP_ID  
FIREBASE_MEASUREMENT_ID=YOUR_FIREBASE_MEASUREMENT_ID  
  
# Stream Configuration  
STREAM_API_KEY=YOUR_STREAM_API_KEY  
STREAM_API_SECRET=YOUR_STREAM_API_SECRET  
  
# AssemblyAI Configuration  
ASSEMBLYAI_API_KEY=YOUR_ASSEMBLYAI_API_KEY
```

Note : to enable running the project locally you can use the following values :

```
SERVER_IP=localhost  
SERVER_PORT=3002  
FIREBASE_API_KEY=AIzaSyB599j7kOPQ9mWNHudx4hh8wdKyI5T-  
i1A  
FIREBASE_AUTH_DOMAIN=finalproject-56ffd.firebaseio.com  
FIREBASE_PROJECT_ID=finalproject-56ffd  
FIREBASE_STORAGE_BUCKET=finalproject-56ffd.appspot.com  
FIREBASE_MESSAGING_SENDER_ID=208463639673  
FIREBASE_APP_ID=1:208463639673:web:de2802c7be88673caf50  
a8  
FIREBASE_MEASUREMENT_ID=G-HV2K04FDW0  
STREAM_API_KEY=rgkeykz9gwms  
STREAM_API_SECRET=pdp3t3ctp7y2qnv2syhrxdta6fq2v2nruaadj  
rqgjync2auwbkrce8bp78a3ym6b  
ASSEMBLYAI_API_KEY=1a5a346f633e470e9f016aa179de9fca
```

4. Open two terminals:

- Run the server: Navigate to the server directory and start the Express server.

```
cd server  
node server.mjs
```

- The server will start and listen for API requests on the specified port
 - Run the client side: Navigate to the root directory and start the Vite development server:

```
npm run dev
```

10.4. Documentation.

This section is for describing each file and its main functionality in the project. (detailed explanations in [appendix](#))

API Folder

- **api/fetchlogs.js**: Retrieves user logs from Firebase Firestore based on a user ID.
- **api/login.js**: Authenticates users by verifying their email and password against Firebase Firestore.
- **api/add-participant.js**: Adds a participant to a meeting by updating the participants list in Firestore.
- **api/meeting-participants.js**: Retrieves the list of participants for a specified meeting from Firestore.
- **api/meeting.js**: Creates a new meeting and adds it to Firestore.
- **api/recording.js**: Handles audio transcription, generates a PDF of the transcription, and updates user logs in Firestore with the transcription URL.
- **api/signup.js**: Handles user signup by creating a new user record in Firestore, hashing the password, and generating a Stream Chat token.

Server Folder

- **Server.mjs**: Configures and initializes an Express server for handling API requests, serving static files, and managing middleware.

Components Directory

- **Home/CreateMeeting.tsx**: Manages the creation and management of video meetings using the Stream Video SDK.
- **Home/Homepage.tsx**: Renders the main landing page, including navigation and options for creating or joining meetings.
- **Home/JoinMeeting.tsx**: Handles joining video calls and managing call interactions.
- **Home/Logs.tsx**: Displays and manages transcription logs, including features for searching, filtering, and pagination.
- **Login/loginPage.tsx**: Renders a login form for user authentication and manages form submission.

- **Signup/Signup.tsx:** Provides a registration form for new users, handling input validation and submission.

10. Resources.

- On speech Recognition Algorithms Shaun V. Ault, Rene J. Perez, Chloe A. Kimble, and Jin Wang, International Journal of Machine Learning and Computing, Vol. 8, No. 6, December 2018.
- A review of Deep Learning Techniques for Speech Processing AMBUJ MEHRISH, Singapore University of Technology and Design, Singapore NAVONIL MAJUMDER, Singapore University of Technology and Design, Singapore RISHABH BHARDWAJ, Singapore University of Technology and Design, Singapore RADA MIHALCEA, University of Michigan, USA SOUJANYA PORIA, Singapore University of Technology and Design, Singapore.
- A Real-Time End-To-end Multilingual Speech Recognition Architecture Javier Gonzales-Dominguez, Member, IEEE, David Eustis, Ignacio Lopez-Moreno, Member, IEEE, Andrew Senior, Senior Member, IEEE, Françoise Beaufays, Senior Member, IEEE, and Pedro J. Moreno, Senior Member, IEEE.
- SPEECH RECOGNITION WITH DEEP RECURRENT NEURAL NETWORKS Alex Graves, Abdel-rahman Mohamed and Geoffrey Hinton Department of Computer Science, University of Toronto.
- Qifeng Zhu, Barry Chen, Nelson Morgan, and Andreas Stolcke, “Tandem connectionist feature extraction of conversational speech recognition,” in International Conference on Machine Learning For Multimodal Interaction, Berlin, Heidelberg, 2005, MLMI’04, pp. 223-231, Springer-Verlag.
- Towards End-to-End Speech Recognition with Recurrent Neural Networks Alex Graves, Google DeepMind London, United Kingdom.
- Using Speech Recognition for Real-Time Captioning and Lecture Transcription in the Classroom Rohit Ranchal, Member, IEEE, Teresa Taber-Doughty, Yiren Guo, Keith Bain, Heather Martin, J. Paul Robinson, Member, IEEE, and Bradley S. Duerstock.
- Trends and developments in automatic speech recognition research Douglas O’Shaughnessy INRS-EMNT (University of Quebec), Montreal, Quebec H5A 1K6, Canada.
- Conformer-1: Robust ASR via Large-Scale Semisupervised Bootstrapping Kevin Zhang* , Luka Chkhetiani* , Francis McCann Ramirez* , Yash Khare, Andrea Vanzo, Michael Liang, Sergio Ramirez Martin, Gabriel Oexle, Roben Bousbib, Taufiquzzman Peyash, Michael Nguyen, Dillon Pulliam, Domenic Donato AssemblyAI Inc.

11. Appendix.

API Folder

api/fetchlogs.js:

- **Purpose:** Retrieves user logs from Firebase Firestore.
 - **Inputs: userId:** User ID from the query parameters.
 - **Outputs**
 - **Success:** List of logs for the user.
 - **Errors:**
 - **400:** Missing userId.
 - **404:** User not found.
 - **500:** Server error.
-

api/login.js:

- **Purpose:** Authenticates users by verifying their email and password against Firebase Firestore.
 - **Inputs:**
 - email: User's email address from the request body.
 - password: User's password from the request body.
 - **Outputs:**
 - **Success:** User ID and authentication token.
 - **Errors:**
 - **400:** Missing email or password.
 - **401:** Invalid email or password.
 - **500:** Server error.
-

api/add-participant.js

- **Purpose:** Adds a participant to a meeting by updating the participants list in Firestore.
 - **Inputs:**
 - **callId:** The ID of the meeting to which the participant will be added.
 - **userId:** The ID of the user to be added as a participant.
 - **Outputs:**
 - **Success:** Confirmation message indicating the participant was added successfully.
 - **Errors:**
 - 400: Missing callId or userId.
 - 404: Meeting with the specified callId not found.
 - 500: Internal server error.
-

api/meeting-participants.js:

- **Purpose:** Retrieves the list of participants for a specified meeting from Firestore.
 - **Inputs:**
 - **callId:** ID of the meeting from the query parameters.
 - **Outputs:**
 - **Success:** List of participants for the specified meeting.
 - **Errors:**
 - 400: Missing callId.
 - 404: Meeting not found.
 - 500: Internal server error.
-

api/meeting.js

- **Purpose:** Creates a new meeting and adds it to Firestore.
 - **Inputs:**
 - callId: The ID of the meeting to be created.
 - userId: The ID of the user to be added as a participant.
 - **Outputs:**
 - **Success:**
 - Status 201 with meeting details including callId, type, and docId.
 - **Errors:**
 - 400: When callId or userId is missing.
 - 500: When an internal server error occurs.
-

api/recording.js

- **Purpose:** Handles the process of transcribing audio, generating a PDF of the transcription, and updating user logs in Firestore with the transcription URL.
- **Inputs:**
 - callId (string): The ID of the call for which the transcription is being processed.
 - url (string): The URL of the audio file to be transcribed.
- **Outputs:**
 - **Success:**
 - Status 201 with a success message and the download URL of the generated PDF.
 - **Errors:**
 - 400: When callId or url is missing.
 - 404: When the meeting with the given callId is not found.

- 500: When the transcript data is invalid or an internal server error occurs.
 - **Throws:**
 - 500: If there is an issue during transcription, PDF creation, file upload, or updating user logs.
-

api/signup.js

- **Purpose:** Handles user signup by creating a new user record in Firestore, hashing the user's password, and generating a Stream Chat token.
- **Inputs:**
 - **req.body.email:** The user's email address.
 - **req.body.password:** The user's password.
- **Outputs:**
 - **Success:**
 - Status 201 with a JSON response containing a success message indicating user creation.
 - **Errors:**
 - 400: When email or password is missing.
 - 500: When an internal server error occurs during user creation, password hashing, or token generation.

Server folder

Server.mjs:

Description:

Sets up and configures an Express server that handles API requests and serves static files.

Features:

1. Initialize Express App:

- Sets up the Express application.
- Configures the server to listen on port 3002.

2. Middleware:

○ **JSON Body Parsing:**

- Parses incoming requests with JSON payloads.

○ **CORS Headers:**

- Configures Cross-Origin Resource Sharing (CORS) to allow requests from any origin.
- Handles preflight requests with OPTIONS method.

3. API Request Routing:

- Dynamically imports API modules based on the request URL.
- Strips query parameters and constructs the file path to the API module.
- Calls the default export of the imported module to handle the request.

4. Static File Serving:

- Serves static files from the dist directory.
- Falls back to serving index.html for any unmatched requests to support single-page applications.

5. 404 Error Handling:

- Handles 404 errors for static files.
- Attempts to serve the requested file or responds with a "Not Found" message.

Functionality:

- **Start Server:**
 - Listens for incoming requests on port 3002.
 - Logs the server's URL to the console.

Usage:

- **Middleware for CORS:** Configures headers to manage cross-origin requests and handle preflight checks.
- **Dynamic API Handling:** Routes API requests to the appropriate file based on the request URL.
- **Static File Handling:** Serves files from the dist directory and supports single-page application routing.

Error Handling:

- Logs errors related to loading API modules.
- Responds with appropriate status codes for errors, including internal server errors and 404 not found.

Home/CreateMeeting.tsx:

- **Description:** This file handles the creation and management of video meetings using the Stream Video SDK. It includes functionality to create a new meeting, rejoin an existing meeting, and display a user interface for managing calls and meeting details.

Component Documentation

1. MyUILayout Component

Description:

- The MyUILayout component provides the layout and controls for managing a video call, including displaying the meeting ID and handling user interactions.

Props:

- **call** (Object): The active video call object.
- **callId** (string): Unique identifier for the call.
- **dialogOpen** (boolean): Boolean indicating if the dialog is open.
- **onCloseDialog** (Function): Callback function to handle dialog close.
- **onSetLoading** (Function): Callback to set loading state.

Returns: React.Element - The rendered UI component.

Methods:

- **fetchParticipants(callId: string): Promise<void>**
 - Fetches participants for a given call ID.
 - **Parameters:**
 - **callId** (string): The call ID to fetch participants for.
 - **Returns:** Promise<void>
- **handleLeaveCall(): Promise<void>**
 - Handles leaving the call, including fetching participants, ending the call, and sending recording data to the server.

- **Returns:** Promise<void>
- **sendRecordingDataToServer(callId: string, url: string): Promise<void>**
 - Sends recording data to the server.
 - **Parameters:**
 - callId (string): The call ID associated with the recording.
 - url (string): The URL of the recording.
 - **Returns:** Promise<void>

2. CreateMeeting Component

Description:

- The CreateMeeting component manages the creation of a new meeting or rejoining an existing meeting. It displays the MyUILayout component with call controls and meeting details.

Returns: React.Element - The rendered UI component.

Methods:

- **sendMeetingDataToServer(meetingId: string, userId: string): Promise<void>**
 - Sends meeting data to the server.
 - **Parameters:**
 - meetingId (string): The meeting ID.
 - userId (string): The user ID.
 - **Returns:** Promise<void>
- **handleCreateMeeting(): Promise<void>**
 - Handles creating or joining a meeting.
 - **Returns:** Promise<void>

Usage:

- The CreateMeeting component initializes or joins a meeting based on the presence of a callId. It handles the creation of a new meeting if no callId is present and manages call state and UI components.

Additional Notes:

- The MyUILayout component is conditionally rendered based on the call state and loading status.
- Error handling is included for fetching participants, sending recordings, and creating/joining meetings.

Home/Homepage.tsx:

- **Description:** This file defines the HomePage component, which serves as the main landing page for the application. It includes navigation, options to create or join meetings, view logs, and information about the project and staff members.

Component Documentation

Methods:

- **handleLogout(): void**
 - Handles user logout by navigating to the home page.
 - **Returns:** void

Returns: React.Element - The rendered UI component of the home page.

Additional Notes:

- The handleLogout function navigates the user to the home page when the logout button is clicked.
- The HomePage component uses the useNavigate hook from react-router-dom for navigation.
- Various sections of the page include informational content and links to other parts of the application.

Home/JoinMeeting.tsx:

Description: This file contains the JoinMeeting component for managing video call interactions. It handles joining calls, sending participant data to the server, and providing UI for call controls and dialogs.

Component Documentation

1. CallControls Component

Description:

Provides the UI elements for managing a video call. Includes controls for toggling audio, video, screen sharing, and cancelling the call.

Props:

- **onLeave:** Function
A callback function that is called when the user leaves the call.

2. JoinMeeting Component

Description:

Handles the process of joining a video call. Manages state for the call ID, the current call, and dialog visibility. Provides functionality for joining a meeting and sending participant data to the server.

State:

- **callId:** string
The ID of the call entered by the user.
- **call:** any | null
The current call object, or null if not in a call.
- **dialogOpen:** boolean
Controls the visibility of the dialog for entering the call ID.

Functions:

- **sendParticipantDataToServer(callId: any, userId: any): Promise<void>**
Sends participant data to the server.

Parameters:

- **callId:** any - The ID of the call.
- **userId:** any - The ID of the user.

Returns: Promise<void>

- **handleJoinMeeting(): Promise<void>**
Handles the process of joining a meeting by creating a new call, joining it, and updating the local storage.

Returns: Promise<void>

- **handleCancel(): void**
Closes the dialog and navigates to the home page.
- **handleLogout(): void**
Navigates to the home page.

3. JoinedMeetingUI Component

Description:

Provides the UI for an active call, including the layout and controls.
Handles leaving the call and showing a dialog to confirm the departure.

State:

- **dialogOpen:** boolean
Controls the visibility of the dialog that appears when leaving the call.

Functions:

- **handleLeaveCall(): void**
Handles leaving the call by showing a dialog and removing the call ID from local storage. Navigates to the home page after a delay.
- **useEffect()**
Listens to the call state and triggers the leave call handler when the call state indicates the user has left.

Hooks:

- **useCallCallingState()**
Provides the current calling state of the call.

Dependencies:

- **CallingState.LEFT**
Represents the state when the call has been left.

Home/Logs.tsx:

Overview

The Logs component is responsible for displaying and managing transcription logs. It includes features such as fetching logs from the server, filtering logs based on user input, and paginating through the logs. It also handles loading states and errors.

Functions

- **fetchLogs**: Fetches logs from the server when the component mounts or the user changes. Handles errors and sets the logs state.
 - **Type**: async function
 - **Returns**: Promise<void>
- **filterLogs**: Filters logs based on the search term and resets to the first page when the search term changes.
 - **Type**: function
 - **Returns**: void
- **handleSearchChange**: Updates the search term state when the user types in the search input.
 - **Type**: function
 - **Parameters**:
 - event (React.ChangeEvent<HTMLInputElement>): The event object from the input change.
 - **Returns**: void
- **handlePageChange**: Updates the current page state to reflect the selected page number.
 - **Type**: function
 - **Parameters**:
 - pageNumber (number): The number of the page to navigate to.
 - **Returns**: void

Login/loginPage.tsx:

Description:

The LoginPage component renders a login form for user authentication. It manages email and password input, handles form submission, and communicates with the server to log in the user.

Details:

- **handleFormSubmit Function:**
 - **Purpose:** Handles form submission, performs validation checks, and communicates with the server to log in the user.
 - **Parameters:**
 - event (React.FormEvent<HTMLFormElement>): The form submission event.
 - **Returns:** Promise<void>

Signup/Singup.tsx:

Description: The SignupPage component provides a registration form for new users. It handles user input, performs validation, and submits the data to the server.

handleSignup Function

Purpose:

Handles form submission, validates inputs, and sends user data to the server.

Parameters:

- **e (React.FormEvent):** Form submission event.

Returns:

Promise<void> - Resolves when the signup process completes.