Quillio

Team 7 — Design Document

Anoop Jain / Ammar Husain / Prashanth Koushik / Charlie Crouse / Jiwon “Daniel” Kim / Jenna Ellis

Table of Contents:

1 Purpose

Note-taking is a standard procedure when participating in lectures, meetings, and study sessions alike, but it’s very easy to leave out a key sentence or to miss out on key details. Quilio makes note of this common problem, and provides a service of transcribing audio from your meeting, parsing the contents, and generating a neat summary based on the main points of your meeting. While there are services currently in place that allow for group contribution in real time, such as Google Docs, none utilize hands-free recording, language processing, and summary technologies to create a comprehensive recap of any meeting.

1.1. Functional Requirements

1. Creating an Organization
   a. Users should be able to create individual accounts.
   b. Organization directors should be able to create organizations and add users to their organizations.
   c. Users in an organization should be able to create meeting groups.
2. Creating Groups and Assigning Permissions
   a. Users with an account in Quillio should be able to create a meeting with other account holders in their organization.
   b. Users should be able to create groups with other members of their organization and then be able to hold meetings with the members of that group. The person that creates the group should be the group

administrator and should be the only person with permission to delete group notes and the group itself. The group administrator can also assign other group members as administrators.

c. Meeting analytics and statistics should only be available to those given access by the creator of the organization.

3. Starting a Meeting and Recording Audio
    a. Users should be able to invite other users to meetings individually.
    b. Users who are a part of a group should be able to start a meeting with all members of that group.
    c. Once initialized, Quilio will automatically record, transcribe, analyze, and summarize meetings. These should be available to the user following the conclusion of the meeting.
4. Saving Meeting Transcripts and Generating Meeting Summaries
    a. Users should be able to view transcripts and summaries for all meetings that they are associated with.
    b. Users should be able to edit meeting transcripts and summaries.
    c. Users should be able to export meeting transcripts and summaries as PDFs.

1.2. Non-Functional Requirements

1. Architecture, Performance, and Technology
    a. The front-end will be built using ReactJS.

  b. The back-end will be written using Flask.
  c. Server-side code will be written in Python and will utilize the Sphinx documentation generator.
  d. NLP will be handled by the Google Cloud Natural Language API.
  e. Deployment will be done using Heroku.

2. Security
  a. The application will enforce authentication and and will protect against common security tests.
  b. The application will restrict learning from corporate users who would like to keep their data secure.
3. Scalability
  a. Heroku will allow the application to effectively handle load-balancing and and scaling.
  b. Processing of audio files and transcript files will be scheduled and performed in the background.
  c. The ability to deploy modularized components through Heroku will allow the application to scale effectively.

2 Design Outline

Quillio will apply the standard client-server model to effectively receive, process, analyze, and store the data created by many users, groups, and organizations.

2.1. Client

The client provides users with a powerful tool that will allow them to navigate their way through the app. Most importantly, this is how they will start organizations, create groups, record meetings, and access their summarized sessions. Utilizing the power of modern front-end web design, the interface will be clean and user-friendly to allow for an convenient, seamless workflow.

2.2. Server

The goal of Quillio is to provide its users with clean, concise transcriptions/summaries of their meetings and other recordings. The main functionality that will draw people to our app is implemented on the server; it is essentially where the magic of Quillio takes place. Recording sessions will be sent from the client to the server to be processed by CMU Sphinx's speech recognition library. The processed recording will then be compressed and stored until it is deemed no longer necessary by the user or eventually by our own recording garbage collection. The processed audio is then fed through our Machine Learning models to generate the best possible summary. Both the processed audio and the summary are the sent to the database so that the user can access them later when necessary.

2.3. Database

We will be using CassandraDB to securely store all the data critical to our app. From user's information to summarized meetings and transcriptions, the database will handle all the requests to retrieve and store. Many optimizations are also easily made available with CassandraDB, so not only will the data store be convenient, but also fast and efficient.

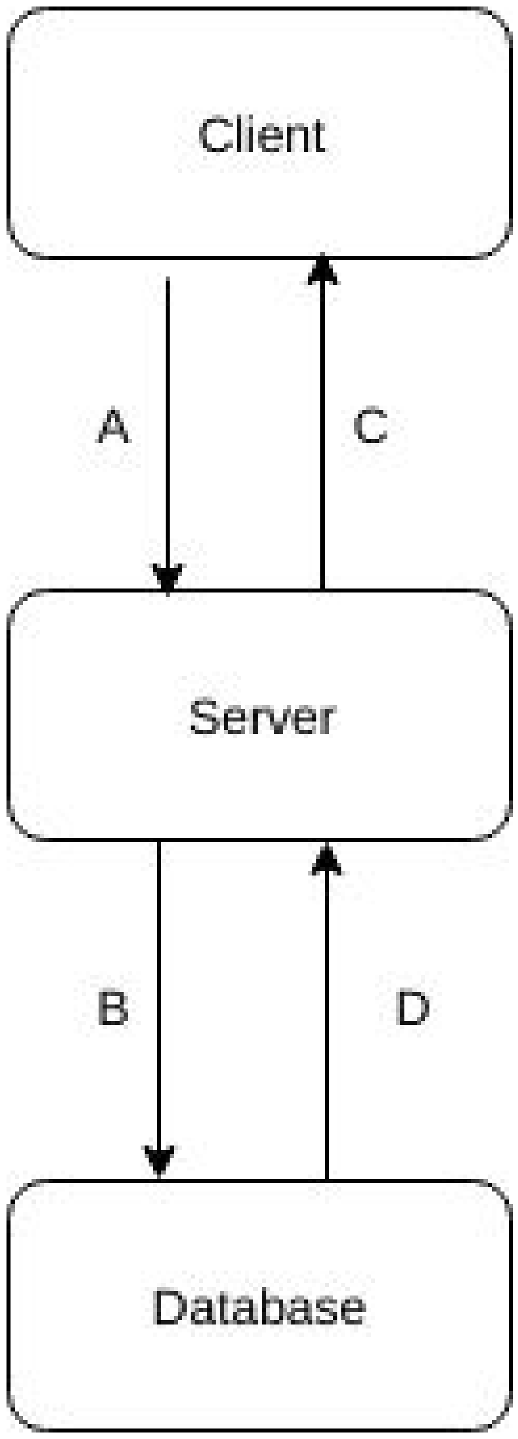


Figure: The above figure is the general communications layout for Quillio: the client communicates with the server to send and receive data. The server sends and receives information to and from the database.

A. Client to Server
   a. Sends recorded audio file from meetings
   b. Sends requests for audio, transcript, or summary
B. Server to Database
   a. Sends requests for audio, transcript, or summary
   b. Transfers compressed audio file to database
   c. Transfers transcripts created from audio files to database
C. Server to Client
   a. Responds to request for audio, transcript, or summary
D. Database to Server
   a. Sends audio files to be compressed
   b. Sends transcripts to be summarized
   c. Responds to request for audio, transcript, or summary

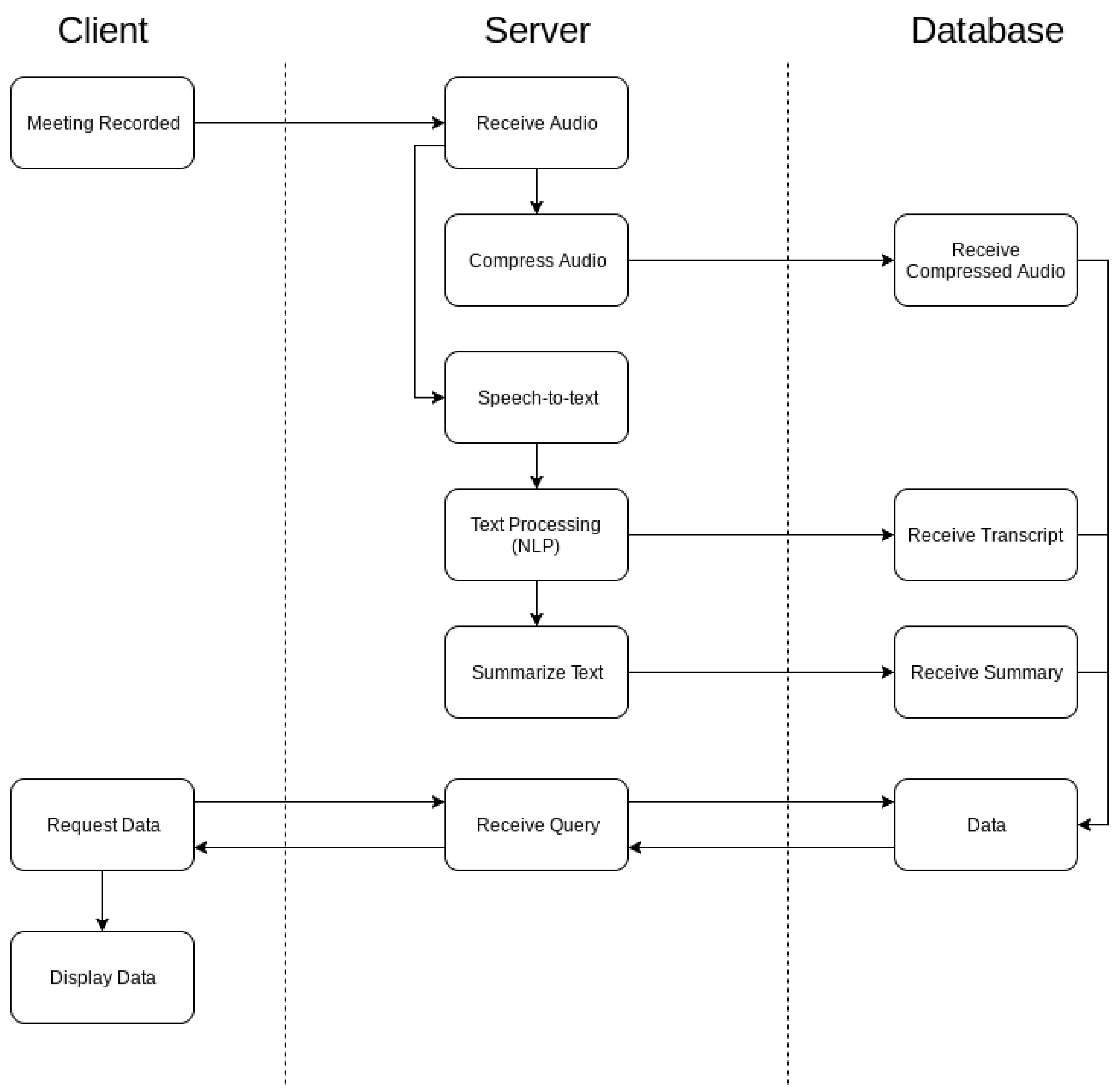


Figure: Diagram of verbose Client-Server-Database description.

3 Design Issues

3.1. Nonfunctional Issues:

Issue: What frontend framework should we use?

I. React
II. AngularJS
III. Ember JS

Of the three frameworks, we ultimately chose React due to its high level flexibility and quick learning curve. Those with front end experience in our group decided that AngularJS presents certain disadvantages to our project ideas, EmberJS had too steep of a learning curve, and React is simple enough to build up on a complex back end system.

Issue: What database software should we use?

I. MongoDB
II. MySQL
III. CassandraDB

We chose CassandraDB because it is very easy to use with ORM (object-relational mapping) structured programs, and it is highly compatible with large data set storage, alike the data that will likely be stored using Quillio. Someone please add more info here because I really don't know why we're using cassandra lol.

Issue: What happens after the audio is processed?

I. Delete it.
II. Store it as a compressed file that the user can access.
III. Keep it in the database that the user can access.

In order to make sure that the storage of audio process doesn't significantly increase the access time of the database, the files will all be compressed before being stored and still accessible by the user if necessary. Eventually, the audio file can be manually deleted, if it is deemed unnecessary. Immediate deletion after transcription could present certain future issues if the user needs the file exactly as it was recorded, or there are issues with the transcription that need resolved based off of the raw recording. Keeping the audio uncompressed in the database will bog down access time to the database over time.

Issue: Which summarizing API should we use?

- **I. SMMRY**
- II. Stremor Automated Summary and Abstract Generator
- III. Intellexer Summarization API
- IV. Build our own summarizing API

SMMRY provides a summarization API that we can integrate seamlessly into our project. SMMRY is an extraction-based summarization algorithm that will extract the most important sentences from transcripts. While SMMRY may not be the best option in terms of summarizing performance, all other options are too costly for this project. The SMMRY API is free for developers as long as they don’t exceed 100 requests per day and the cost of exceeding this limit is much lower than the base costs for other APIs. The cost to build our own summarization API using NLP toolkits such NLTK would be zero, but the benefits of doing this don’t justify the time that our group would have to put into the development process.

3.2. Functional Issues:

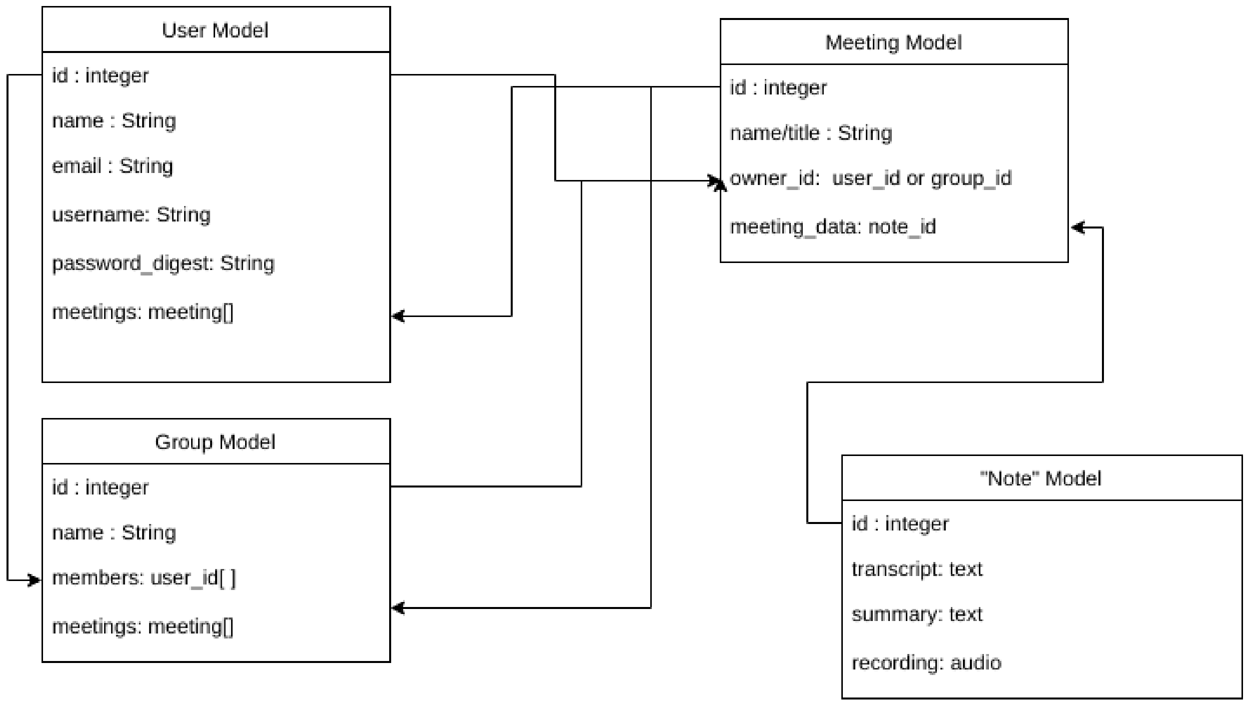
Issue: How can we handle administrative roles within meeting/note groups?

- I. Group administrators
- II. **Public and Private meetings**
- III. Individual User meeting ownership

This issue can be resolved by making public meetings accessible to an entire group of say 20 people, even if the actual meeting and produced notes only took place with three people of a group. A private meeting would be of the same scenario, but the meeting notes would not be accessible by any other members of the group. Group administrator designation and individual user meeting ownership would both complicate the access rights of meetings designated to organizational groups.

4 Design Details

4.1. Database Schema



4.2. Description of Classes and Database Tables

The above diagram is a mockup of our planned database schema. For the time being, we see this design as a feasible means of organizing all the data necessary for our product to function accurately and efficiently. Each arrow indicates a one way relationship between models.

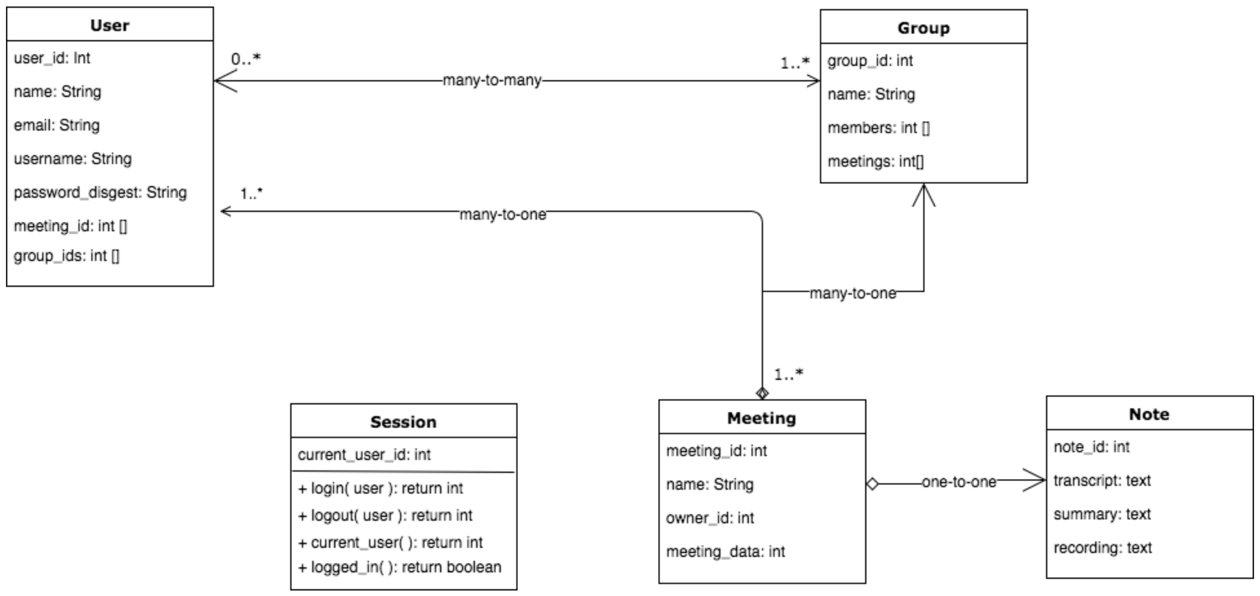
1. User
    a. All users that have an account with Quillio are represented by this model.
    b. Contains all vital information about a user, such as name, email, password, etc.
    c. All meetings that a user has access to is also stored within the model.
    d. All string values associated with the user model are encrypted before being stored in the database.
2. Group
    a. A group is a collection of users that need to meet on a regular basis. This model exists to eliminate time wasted in creating the same set of access to meetings for the same people repeatedly.
    b. A group can be named in order to have easy identification from the user side, and this name will be stored as an encrypted string.
    c. A group has a field that holds all user id’s that are a member of the group.
    d. A group’s collection of meetings can be accessed through a group model.
3. Meeting
    a. A meeting is a “parent” model of the data collected when creating and recording a meeting. The meeting model encapsulates the transcript and summary data, and stores it alongside the group_id or the “owner_id”, which is the single user that owns the meeting.
    b. The name field of a meeting signifies a topic or purpose, and is stored as an encrypted string.

   c. The meeting data field is an integer field that corresponds to the id of a note model, described below.

4. Note
   a. A note is the group of data that makes up all important information collected during a meeting.
   b. The transcript (the cleaned script from the recording) is stored as encrypted text.
   c. The summary (output from a summary API when given the transcript) is stored as encrypted text.
   d. The recording is stored in the database as a compressed audio file for user reference if necessary.

4.3. Class Diagram

As stated previously, Quillio is dependent upon an Object Relational Mapping (ORM) structure. The database schema given previously is very similar to the following class diagram, which contains additional object structures and more details to class relationships. The additional classes are not saved in the database and exist as supporting data organization classes.



4.4. Speech to text

For speech to text, we will be using the CMU Sphinx to process the audio files into text. If one mic is being used, we will using watson's speech diarization api to separate the transcripts into different speaker labels. Watson's API comes with some tuning parameters such as keyword specification and keyword snapping threshold.

4.5. Transcript cleanup

Once we get a transcript, there will most likely be text that is processed incorrectly. This subcomponent will be parsing the transcripts and cleaning up any words or phrases that may erroneous. Watson's speech to text will be able to tell us the accuracy for each word, which we can use to implement a minimum cap.

4.6. Text summarization

To summarize text, as our POC, we will be using the SMMRY API to produce an extraction based summary. These will basically be highlights from the conversation, as no abstraction is generated. However in further sprints, we will be implementing our own conversational text summarization scheme that can be customized to work efficiently with conversation transcripts rather than other types of texts.

4.7. Transcript tagging

Given a cleaned transcript, will will have to extract keywords that represent topics covered by the meeting, if an agenda has not been provided. We will be using LDA algorithms to extract keywords and generate a word cloud for the UI, and store these tags to query in the future, and snap further meeting notes to existing tags.

4.8. Email API

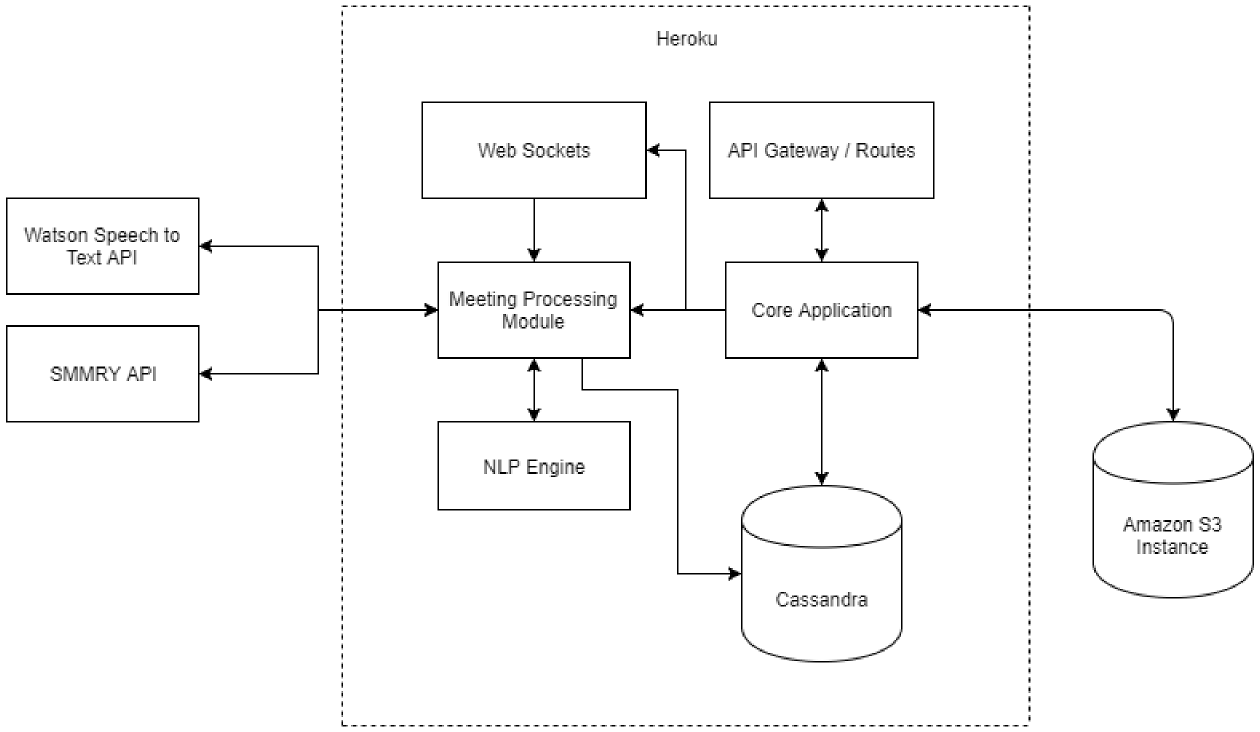
We will use the SendGrid email API to send email notifications and files to Quillio users. Email notifications are necessary to remind users for upcoming meetings, deliver summary documents to people who could not make a meeting, and other regular security checks with user accounts. The SendGrid API integrates as a Heroku deployment add on and is free to developers and in production when emails are sent in low volumes. SendGrid is also very flexible with its API customization features and integration with different tech stacks.

4.9. Datastore

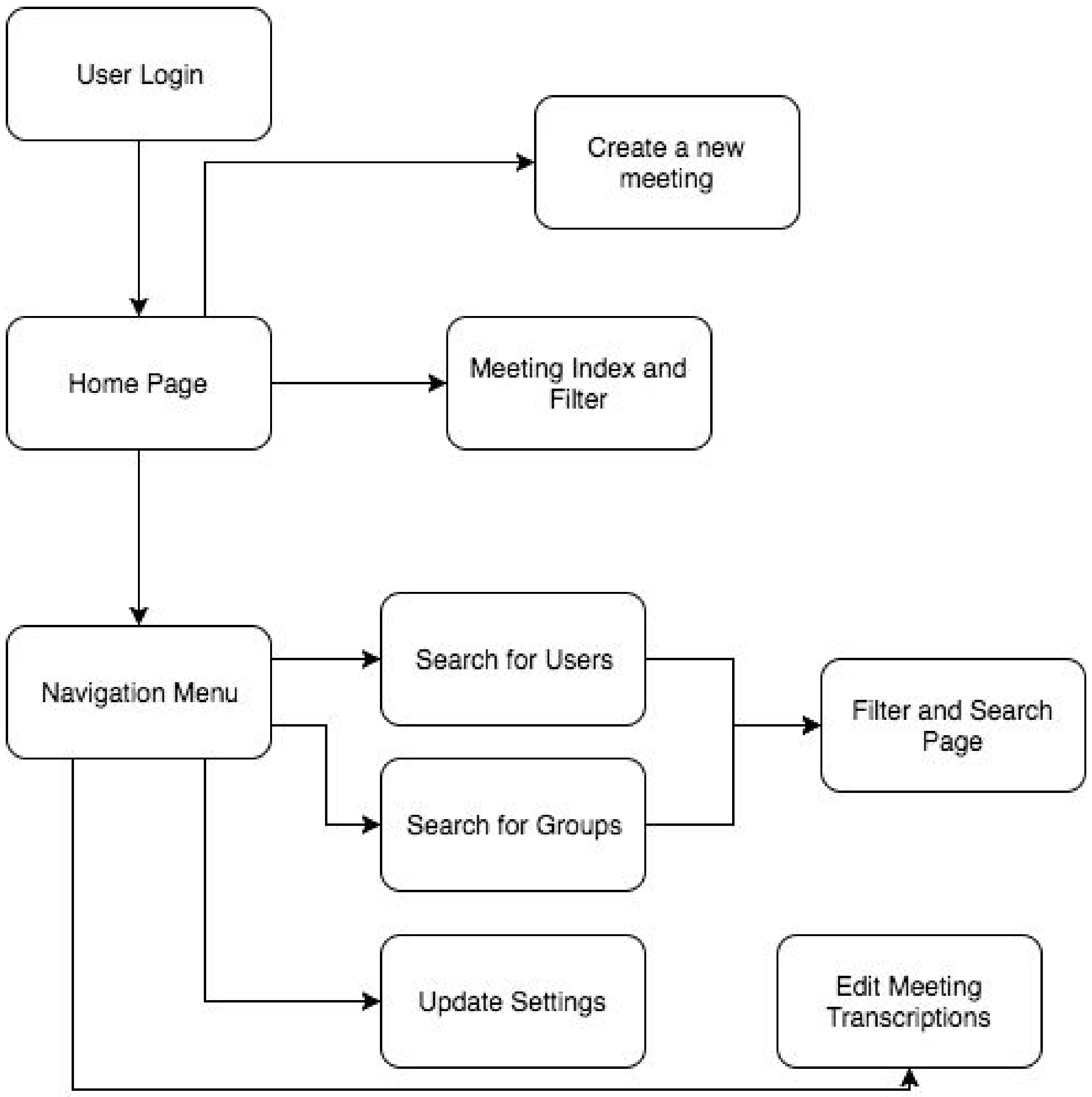
We will need 2 different kinds of datastores. Cassandra will store all of our application data, from users and notes, to sessions and processes. However if we want to store the original audio files as well, we will need have possibly an Amazon S3 deployment for object storage.

4.10. Deployment

We will be deploying our project to production using Heroku microservices. Heroku's easy integration with emailing services, GitHub, and various other add ons make it ideal to build out a large scale web application project alike Quillio.



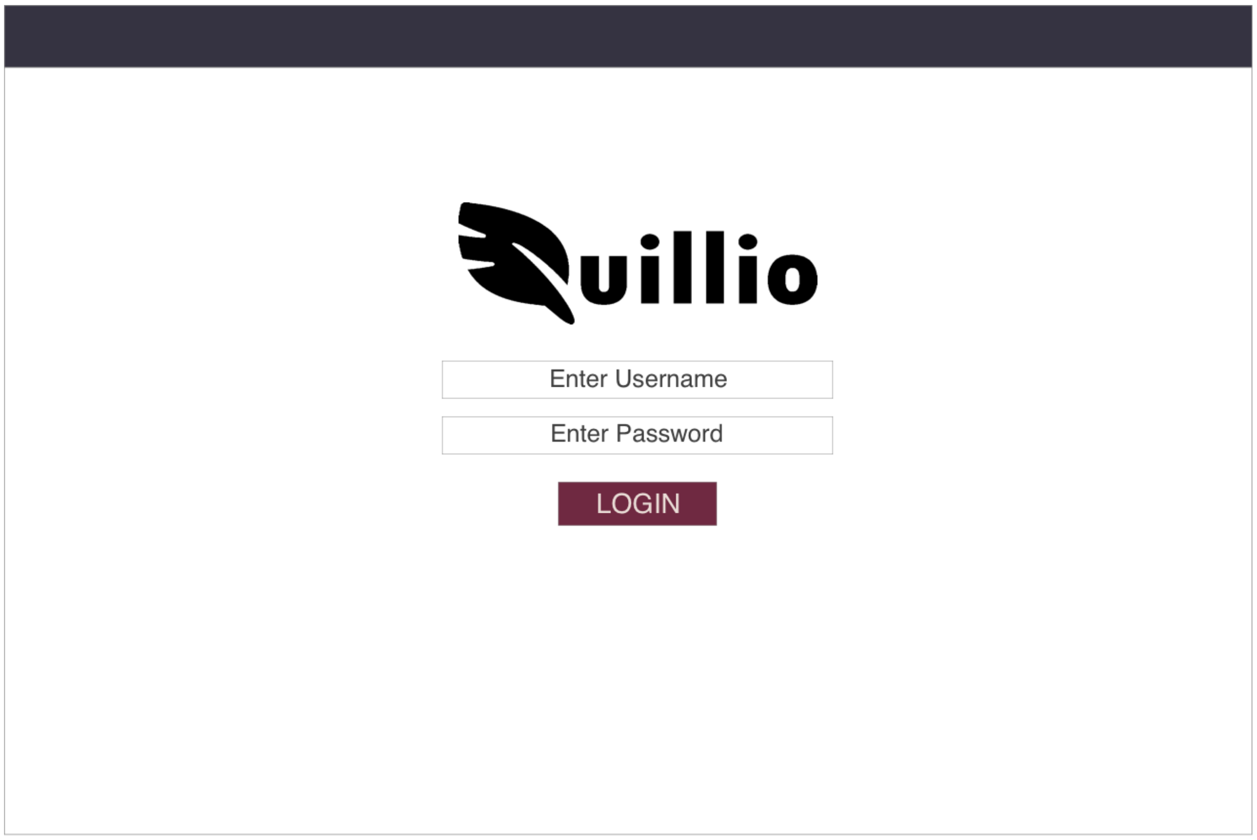
4.11. Navigation Flow



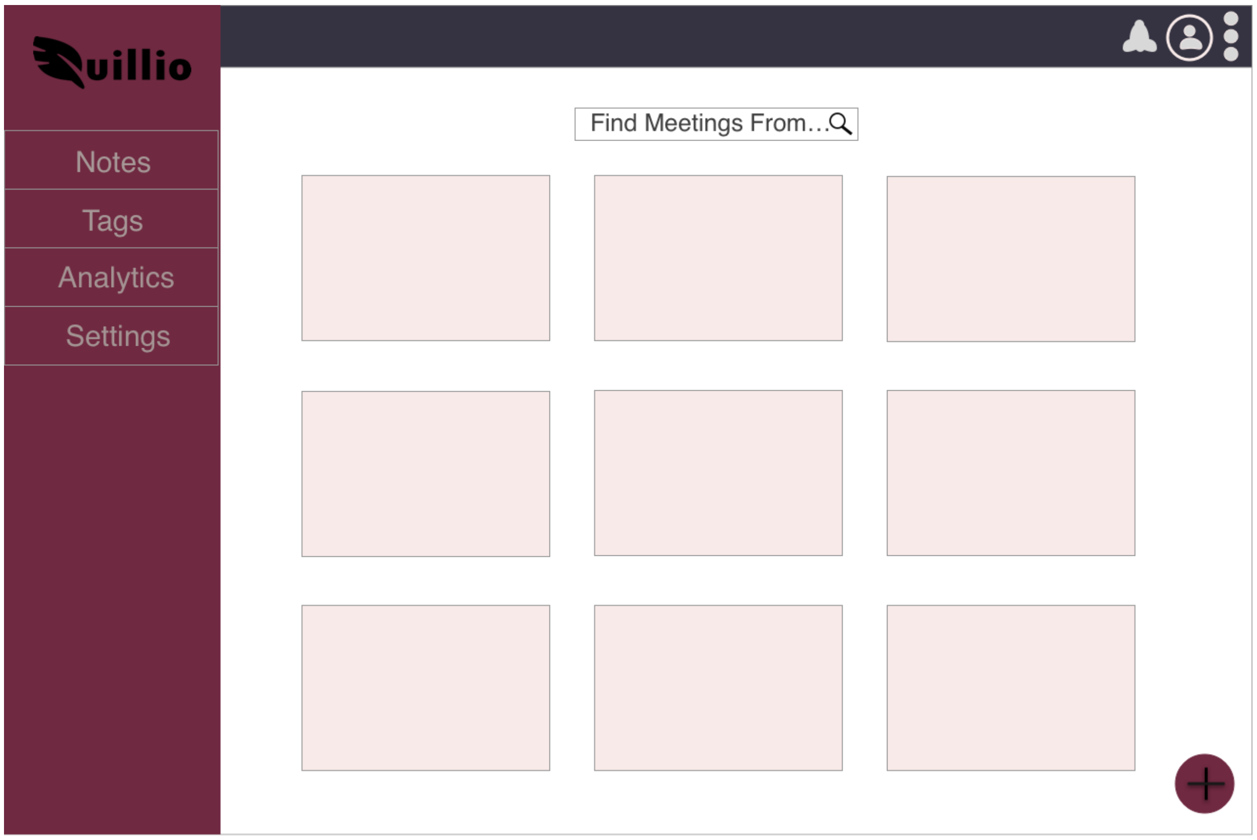
The above diagram is a flowchart of user navigation options upon visiting the Quillio login portal. The user logs in, and is redirected to their home page. From the home page, the use has three options for navigation or activities. The user can create a new meeting, search and filter through all of the meetings that they have participated in thus far, or use the navigation menu.
The navigation menu has options for the user to search for other Quillio users and groups, update their personal settings, and edit their existing meeting transcripts.

5 User Interface

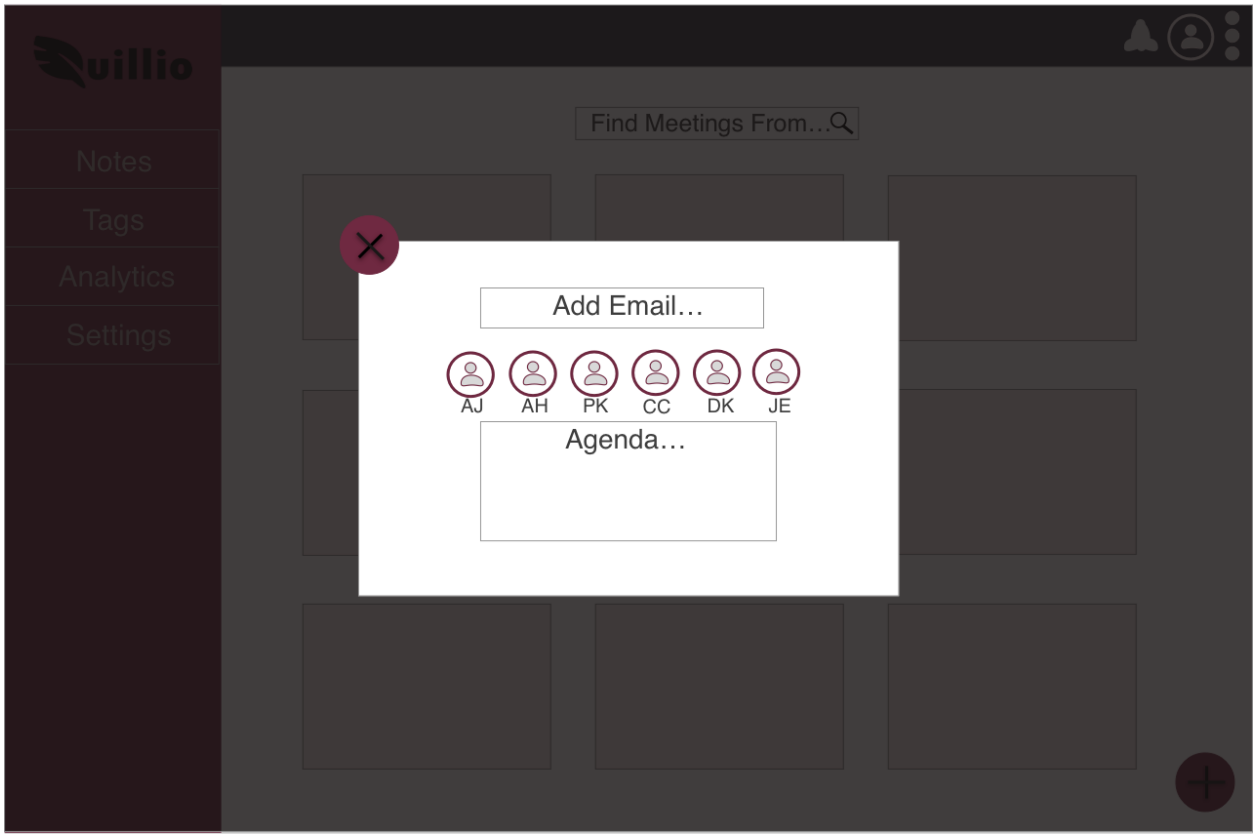
We have designed the UI for the application to be fairly consistent across various navigation hubs. We will be using Material Design Bootstrap to add some additional front end features that would normally be difficult to implement. We will be using ReactJS to build out the front end, along with handlebars templating to build out the views in a simple manner.
The major components of the UI are found in the sidebar menu, the central “hub” of directories a person can see upon logging in, and the header bar above it.



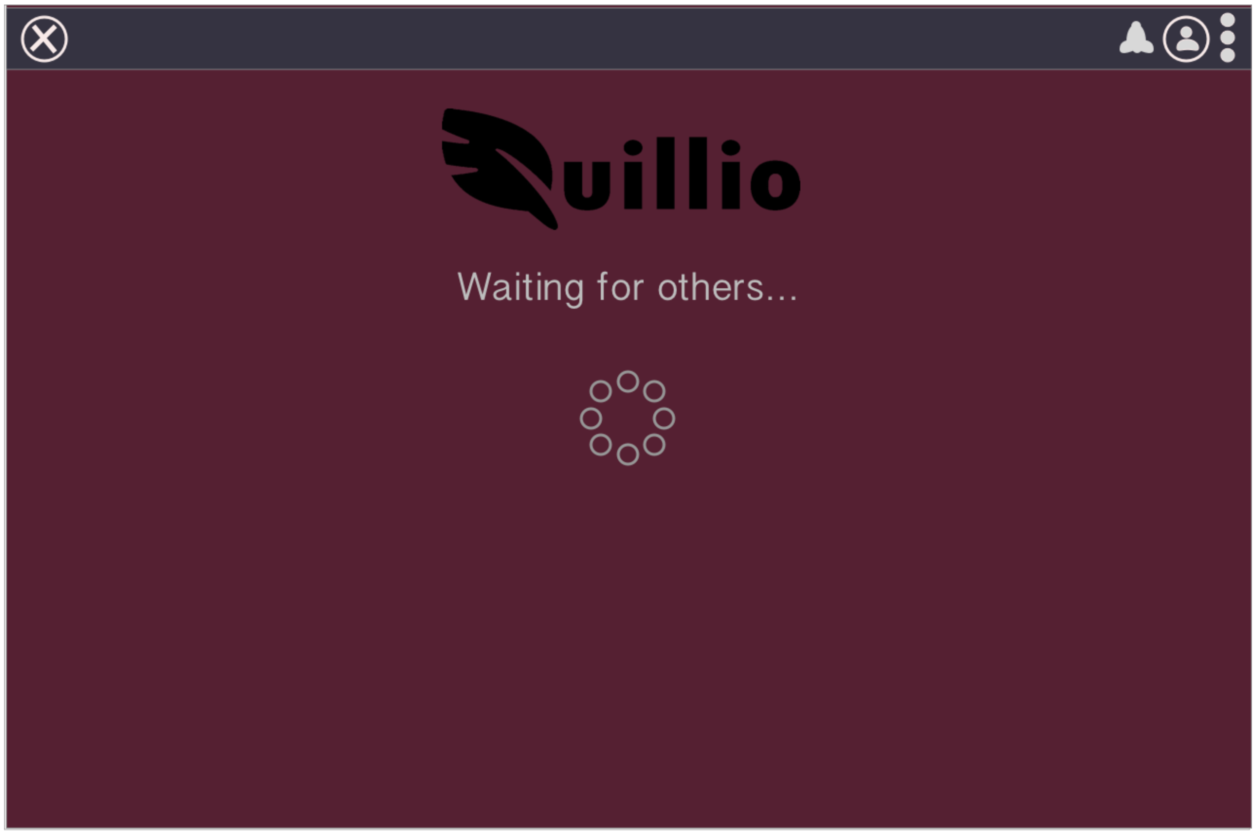
The desktop login page frame.



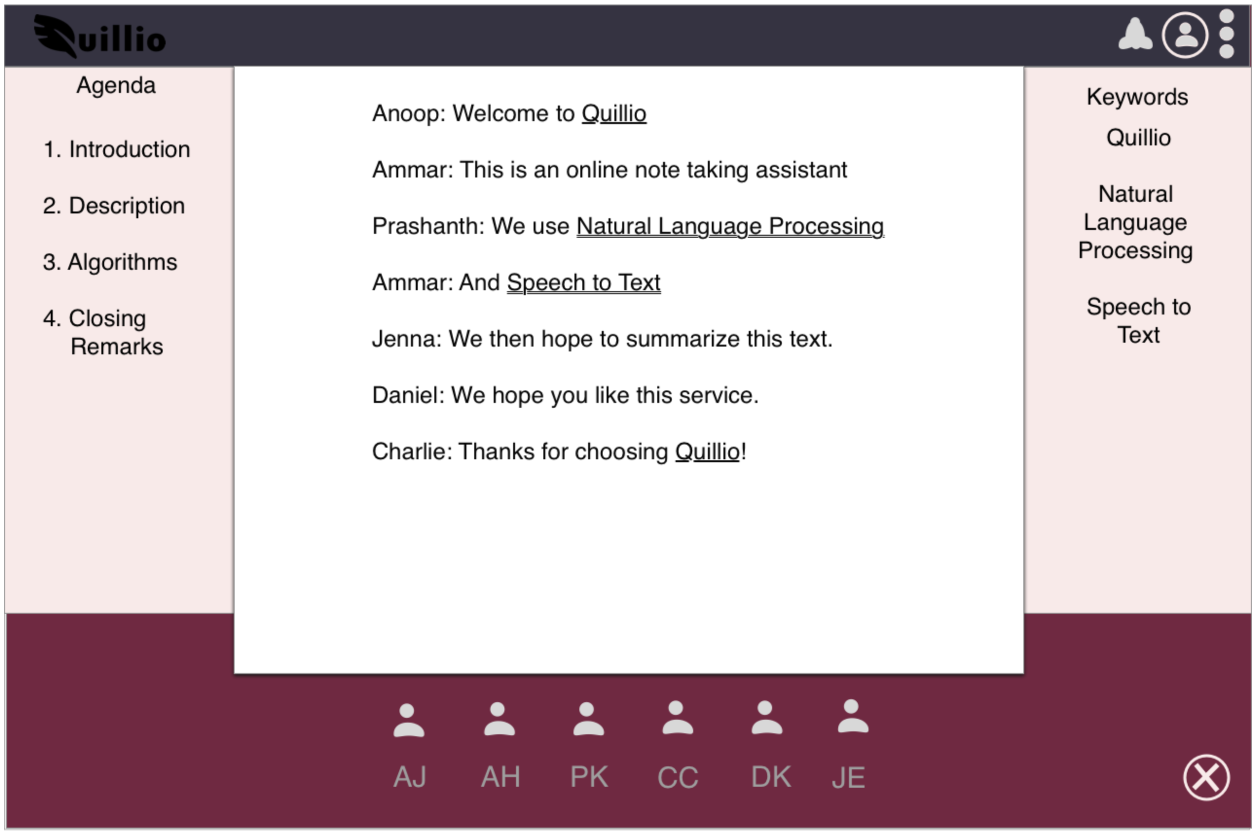
This is the page that renders upon a user logging into their account. The objects listed in the center of the window are all the meetings a user has access to so far. The sidebar is a navigation element, in which a user can select one tab to do an action, such as, search for another user or group, update their settings, edit transcripts and notes, and manage their meetings.



Upon clicking the "create a new meeting" button in the bottom right corner of the home page listed previously, this modal will render. The user will type in the emails of other Quillio users to add them to a meeting. The user can also add an optional agenda for the meeting so that everyone knows what they are going to talk about.



Once a meeting has been created and all invited users will be shown this page up until all invited members have joined the meeting.



Anoop: Welcome to Quillio

Ammar: This is an online note taking assistant

Prashanth: We use Natural Language Processing

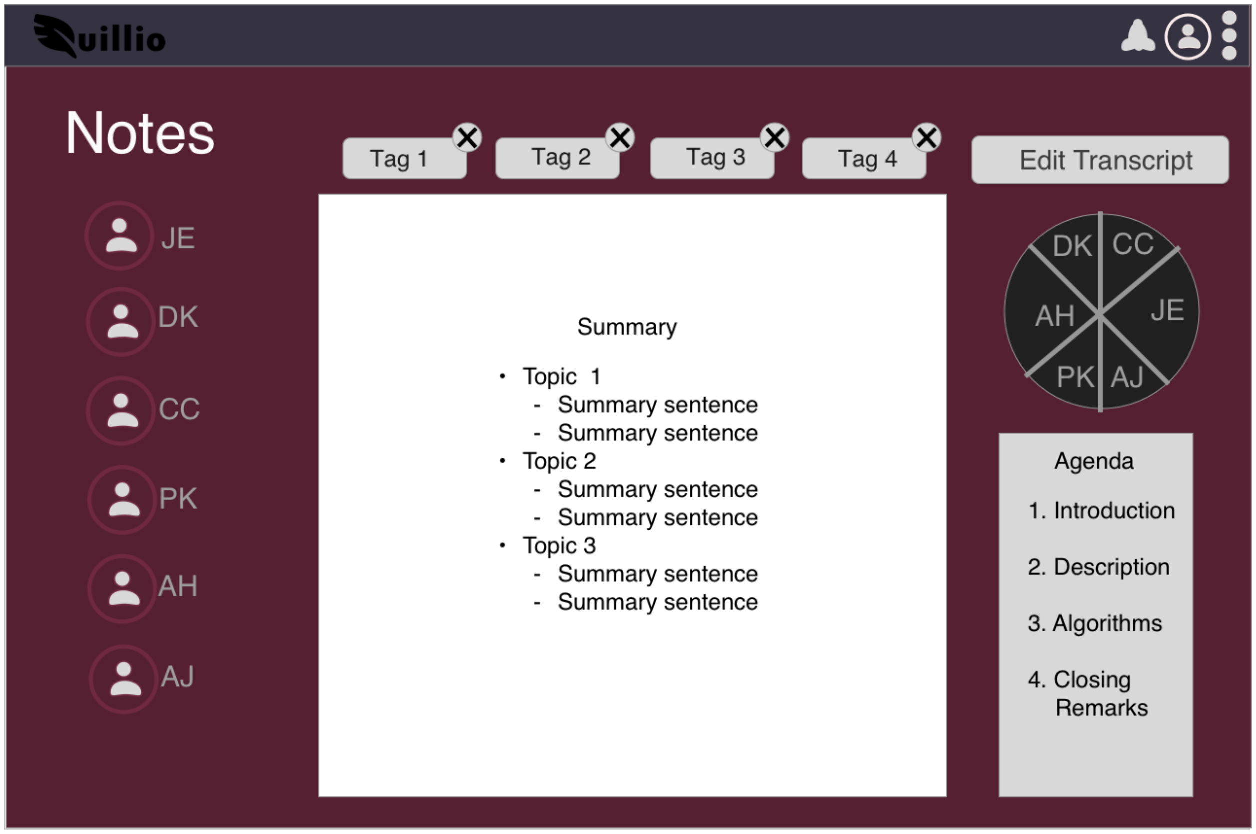
Ammar: And Speech to Text

Jenna: We then hope to summarize this text.

Daniel: We hope you like this service.

Charlie: Thanks for choosing Quillio!

Once all users have joined the meeting, the meeting can begin. This page will show a live transcription of the meeting, the agenda of the meeting, and keywords/important topics of the meeting. Users will also be able to see who else is participating in the meeting or exit the meeting at any time they'd like.



Summary

- Topic 1
  - Summary sentence
  - Summary sentence
- Topic 2
  - Summary sentence
  - Summary sentence
- Topic 3
  - Summary sentence
  - Summary sentence

Agenda

1. Introduction
2. Description
3. Algorithms
4. Closing Remarks

This screen shows how users can access the transcripts or summaries of previous meetings. This page includes the different keywords and agendas of the meeting. Users have the option to view or edit the transcript. We have also a pie chart so administrators can see how much each person contributes to the discussion.