VoteHub: Highly Customizable and Secured Voting System for Institutions Providing AI-Based Transcript Generation

Norbert Biró
Babeș-Bolyai University
Cluj-Napoca, Romania
bironorbi0356@gmail.com

Gergő Kelemen
Babeș-Bolyai University
Cluj-Napoca, Romania
zskelemen73@gmail.com

Dénes Bálint Kelemen-Fehér
Codespring
Cluj-Napoca, Romania
kelemen.balint@codespring.ro

Katalin Rácz-Nagy
Codespring
Cluj-Napoca, Romania
nagy.katalin@codespring.ro

Károly Simon
Babeș-Bolyai University
Cluj-Napoca, Romania
karoly.simon@ubbcluj.ro

***Abstract*—Institutions often face challenges in managing the voting process efficiently and securely during internal meetings. In large groups, voting by show of hands is slow and prone to errors, while many digital solutions lack flexibility, real-time feedback, and secure, organized result storage.**

This paper presents a system which aims to provide a reliable platform for institutions to organize meetings and manage the voting process on different topics during these meetings. Institutions are organized into closed forums. The system distinguishes between two roles: moderators manage the list of members, forums, meetings, and topics; forum members can participate and vote in meetings. The system also provides the ability to automatically create transcripts from the audio recording of a meeting.

***Index Terms*—Voting System, Software Architecture, Secure Forums, AI-Based Transcript Generation**

I. Introduction

The VoteHub application provides a platform that makes it easy, convenient and secure to organize institutional meetings where participants vote on various topics. Each institution has its own solution regarding the voting processes applied at its meetings. One obvious solution is to vote by show of hands, but this process is time-consuming in larger groups, as it involves counting the votes individually, then calculating the final result and it also can be prone to errors. Some software solutions and tools are also frequently used, such as Google Forms or Microsoft Forms. In these applications, polls and voting forms can be created and shared among members of the institution. However, in a longer meeting, this approach is not optimal either, as there is no live feedback and no way to track the current state of the vote. There are also some specialized systems, some of them including hardware elements, but these solutions are too rigid and not customizable enough for a wider acceptance by institutions.

VoteHub provides a more flexible solution that simplifies the organization of meetings and the management of voting processes, while at the same time makes the process fully customizable. Institutional meetings are organized into closed forums, where access is by invitation only, ensuring that only authorized users have access to sensitive information. In addition, the application supports online and hybrid meetings, where the status of a meeting can be followed in real time and users can actively participate in the voting process. It is also possible to automatically generate reports and transcripts based on the meetings.

The paper describes the different aspects of the application, such as the architecture of the system, the functionalities, the tools and technologies used. First, a general overview of the system architecture and its functionality is presented. Then, the technological solutions by which the system achieves these functionalities is discussed in detail. The paper will be concluded by listing some possibilities for further development.

II. Functionalities

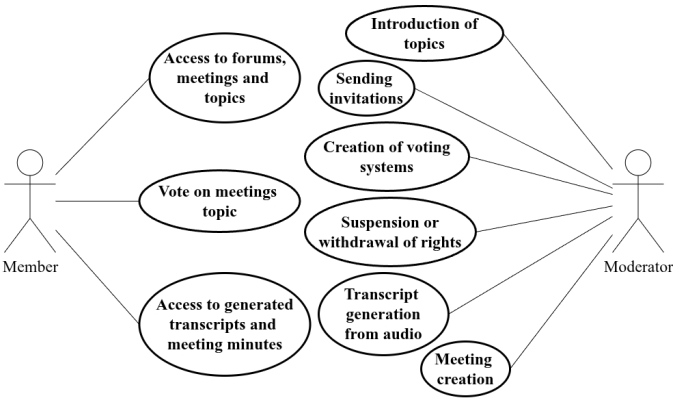


Fig. 1: Use cases of VoteHub application.

The system distinguishes between two user roles: member and moderator. Figure 1. shows the use-cases of the application. The two different roles are introduced because of the different privileges. Users can receive privileges through invitations, a person can receive forum invitations for both roles. Users can choose to accept or decline the invitation. Any registered user can create a forum, in which case they automatically have moderator rights within the created forum.

Main functions of moderators:

- Create meetings and topics. The central element of the application is the forum, and the management of meetings and topics within the forum is one of the main tasks of moderators.

- Send invitations to other users. Invitations are sent by email. The invited person does not need to be registered in advance. When inviting a user, their role must be defined.
- Edit information of forum and its meetings and topics. This includes editing entity names, descriptions, scheduled dates.
- Create voting systems. The voting system defines the voting process that is used to determine the outcome of the vote.
- Suspend or revoke the privileges of other users in the forum. A moderator can temporarily suspend or completely exclude members and even other moderators from the forum. They can also revoke the voting rights of members on a given topic (in this case, there is no exclusion from the forum).
- Conduct meetings. Only one moderator can start a meeting. Once the meeting has started, the moderator can set a topic to be voted on. The moderator can see the number of members who have joined and the number of votes. The moderator is responsible for starting and closing the votes. Once a vote is closed, the result is displayed, showing which members voted for which option.
- Generate transcript from audio recording. Based on the audio recorded during the meeting, moderators can generate a transcript that can be edited later by them and shared with other users.

Main functions of members:

- View forums, meetings, topics.
- Participate in meetings.
- Have voting rights. Their votes determine the outcome of the topic.
- They can access the meeting transcript, which is available for download and sharing.

III. Voting system and result calculation

To determine the results of a ballot, the following criteria are taken into account:

1) Quorum, which is the percentage of members that must be present to constitute a valid vote.
2) The percentage of votes required for a voting option to be the winning option. The system allows moderators to specify whether the percentage of votes cast for an option should be calculated from the number of participants present or from the total number of members in the forum.
3) Whether abstention is accepted as a valid voting option.

The moderator can create voting systems from templates, modifying the values of the templates, or define a completely new system. For voting systems created from templates, by default abstention is not a valid option and the percentage of votes for options will be calculated from the number of members present. Once topics are created within a meeting, the moderator can assign voting systems to these topics, and the outcome of each topic will be calculated according to the criteria of the assigned voting system.

The calculation of the results starts with determining whether results are calculated from the number of forum members, or from the number of participants present. This number will be denoted by T_0. Then, the number of votes cast (a_i) will be calculated for each option. From this we obtain the percentage of votes on an option by $p_i = a_i/T_0$. Then, if participants have the ability to abstain according to the voting system, the number of abstentions A is subtracted from the number of votes cast, i.e. $T_1 = T_0 - A$. Next, we check the previously mentioned criteria. A vote is not considered successful if there are fewer members present than the minimum number of votes specified in the 1. criterion, if $T_1 = 0$, or if there are two options which have the same number of votes. In all other cases, the winning option is the one for which p_i is greater than the value specified in the 2. criterion. If the abstain option happens to be the winning option, then the ballot is again not considered successful.

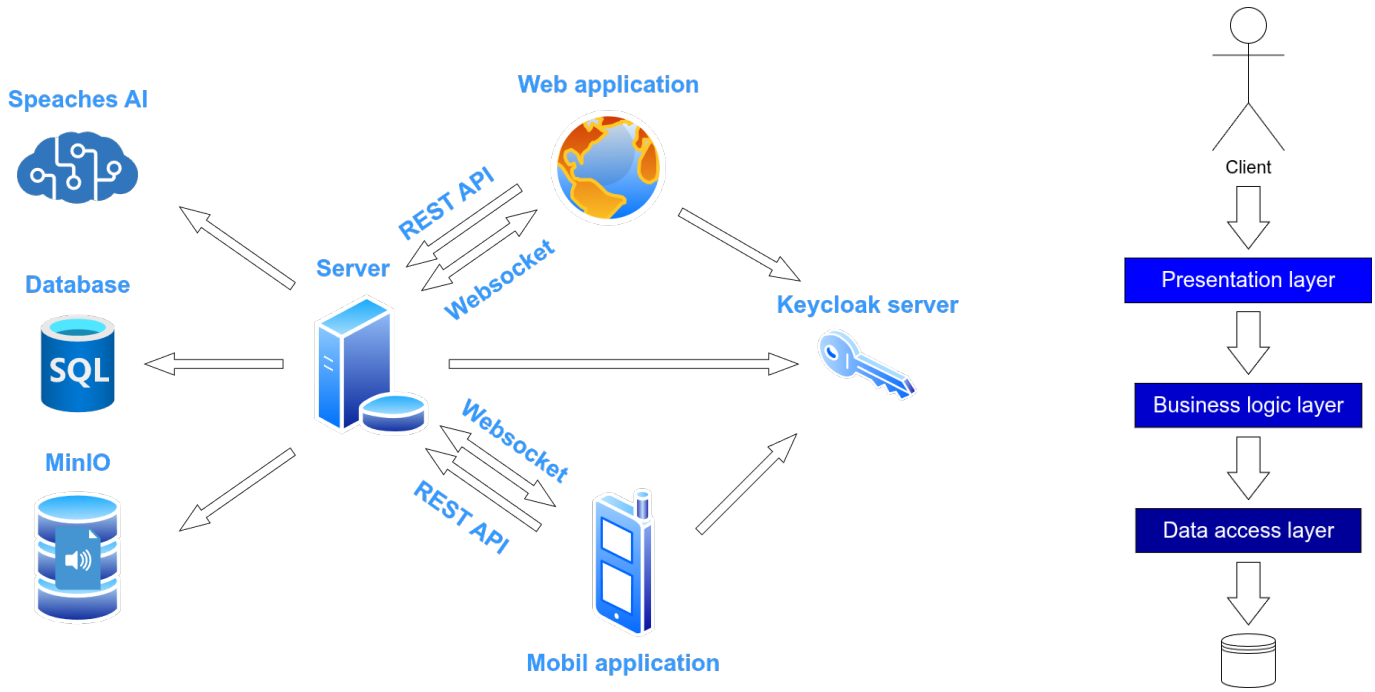
The system provides templates for the more common types of voting, which are:

- Simple majority - $50\%+1$ vote for an option to win ($p_i = 0.5$)
- Two-thirds majority - $2/3$ of members must vote for an option to win ($p_i = 0.(66)$)
- Full consensus - all members must vote for the same option ($p_i = 1$)

IV. General architecture

Figure 2(a). shows the architecture of the VoteHub system, which consists of three main services: a web and a mobile application, served by a central server.

- Server - The server provides a RESTful API and is written in Kotlin using the Spring framework.
- Database - The database of the system is a MySQL database. Database migration is done using Liquibase.
- Artificial Intelligence-based Transcription Service - The system uses Speaches AI, based on the OpenAI Whisper general-purpose speech recognition model, to transcribe audio material recorded during meetings.
- File-based storage system - The MinIO service is responsible for the storage of files. All audio files that arrive for transcription are stored here and are available to users.
- Web client - The web client provides an interface for moderators to perform administrative tasks. The application is written in React using Typescript, state management is implemented using Redux, and the UI elements come from the MaterialUI design library.
- Mobile client - The mobile client allows members to vote. The application is written in React-Native, the code is compiled and packaged using Expo, and the UI elements are built using the React-Native Paper library.
- Keycloak - The identity provider used for user authentication is Keycloak. The Keycloak server allows users to verify their identity based on JWT tokens [1].



(a) Architecture of VoteHub.

(b) The three-layered architecture of the server.

Fig. 2: General and Server Architecture

V. Server

The core of the VoteHub system is the server, which stores data, performs the necessary operations to implement the business logic, serves the web and the mobile clients, and handles audio recordings, transcriptions.

A. REST API

Client-server communication is done via a RESTful API in JSON and Multipart format. This is used for data retrieval, data modification and other functionalities. A DTO (Data Transfer Object) design pattern is applied, which allows communication to be independent of the database schema and the system models [2].

B. Websocket communication

During the use of the web and mobile applications, clients may want to receive real-time notifications about changes in system data, which cannot be achieved through the REST API. To enable two-way communication for transmitting and receiving real-time data changes, the Server communicates with clients via WebSocket [3].

When members join a meeting, they are automatically subscribed to the channel associated with the meeting, where they receive information about the meeting's status. A meeting can have multiple states set by the moderator. The mobile client reacts in real-time to these changes, displaying them to the members with every state change, which may include starting the meeting, switching topic, stopping the vote, or ending the meeting. The Server sends notifications to the connected clients about these changes. Additionally, when members join, leave, or cast a vote, the Server broadcasts this information on another channel to the moderators. In this way, the number of attendees and the number of votes cast can be tracked accurately and in real-time.

The processing of recorded audio files can also have multiple states. After uploading an audio file, moderators receive updates via WebSocket about the current processing stage.

C. Architecture

The server architecture, shown in 2(b), consists of three layers: Presentation Layer, Business Logic Layer, and Data Access Layer. Organizing functionalities into layers eliminates unnecessary dependencies between components, ensuring modularity [4]. The implementation of one layer can be replaced without modifying the other layers. Each layer has a well-defined set of responsibilities. Communication between the layers always follows the path from the Presentation Layer through the Business Logic Layer to the Data Access Layer. The services provided by a layer are defined in Kotlin interfaces.

The Presentation Layer provides the opportunity for the web and mobile clients to make requests to the server. The VoteHub server provides a RESTful API, allowing clients to communicate using HTTP requests. This API is implemented by classes annotated with the @RestController annotation, which specifies that the given class is capable of handling HTTP requests.

The Business Logic Layer defines the entire functionality of the server. After receiving a request, the RestControllers forward it to the appropriate Service class. Classes that implement Service interfaces are annotated with @Service.

The classes located in the Data Access Layer establish a connection with the MySQL database. The interfaces inheriting from the JpaRepository interface are annotated with @Repository.

D. Transcript Generation and meeting minutes editing

Moderators can generate transcripts from the audio recordings of meetings. The tool used for this feature is Speaches AI, which utilizes OpenAI Whisper-based artificial intelligence models.

The transcripts are stored in the database in a dedicated table, which contains the transcript text, the processing status, and a foreign key reference to the corresponding meeting.

After the meeting ended, moderators can upload an audio file with a .mp3 or .wav extension. The language of the audio file must also be specified to ensure more accurate processing. The list of supported languages by the chosen model is available via the REST API. The related service stores the audio files to be processed in a queue. Moderators can edit the generated transcript, which will then be marked accordingly. It is also possible to delete the generated transcript, even if an error has occurred.

A transcript can be created not only by uploading an audio file but also by manually entering the text.

E. Storing Audio Recordings

As mentioned in V-D paragraph, moderators can upload audio recordings to generate transcripts. To ensure future access, these recordings are stored using MinIO, an open-source object storage system compatible with the Amazon S3 API. Communication between the Server and MinIO is handled by the MinioClient class from the io.minio:minio library, enabling basic operations like saving, deleting, and retrieving files, as well as managing buckets.

The system also stores audio metadata (filename and language) in the MySQL database.

A dedicated service manages MinIO interactions. Moderators upload files as multipart/form-data, which will be saved in a bucket named forum-forum_id. Each meeting has a subfolder named meeting/meeting_id/audio, where audio files are saved with a randomly generated name and the original extension. This structure supports multiple recordings per meeting.

When a transcript is deleted, both the database entry and the corresponding file in MinIO are removed.

VI. Web and mobile application

A. Authentication

When the application is first opened, the user sees an introductory screen, where by clicking the Login button, the application redirects them to a page, where they can login or register. When registering, the user must verify their email address. These processes are performed by the Keycloak server. If the login is successful, the Keycloak server generates an access token and a refresh token, and then redirects the user back to the application. These tokens are then saved on the users device (or the browsers local storage, in the case of the web application), and until these tokens are valid, the user does not have to sign in again.

The communication with the Keycloak server happens according to the OIDC protocol, and thus the generated tokens are used based on the OIDC specifications [5]. If the user wants to access a protected resource, the application automatically appends the access token as a bearer token to the request header sent to the server. The server then checks this and sends the appropriate resource only if the token is valid. Access tokens have an expiration time which is rather short, and in order to avoid the user having to log in again, a refresh token is used to obtain a new access token. If the user is trying to access a protected resource, and no access token is present in the request header, the server responds with a 401 status code. If the user is logged in, but does not have permission to view the resource, then the server responds with a 403 status code.

B. Meeting management in the web application

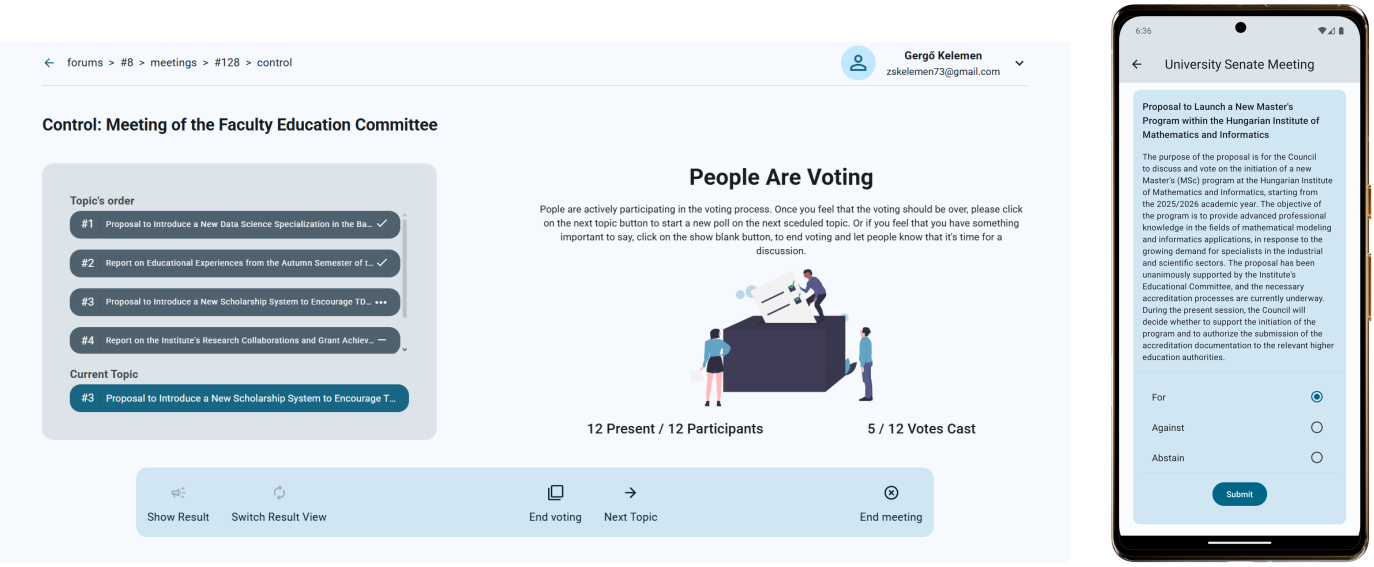
The primary purpose of the web application is to provide an administrative interface for moderators. In this section the process of how moderators can manage a meeting will be presented.

A separate page is available to moderators in order to control meetings. A single moderator has to conduct a meeting, from its start to its end. Before the meeting is started, the moderator can only see a text on this page, which informs him that the meeting has not yet been started. A start button is also visible below the text, which can be used to start the meeting.

By clicking the start button, the meeting starts and the meeting control interface appears, which consists of the following three parts:

- On the left side of the screen is a list that shows the topics of the meeting and below that the topic which is currently under discussion, if any.
- At the bottom of the screen is the control panel itself, where by clicking on various buttons, the moderator can start or stop voting, or view the results of a vote.
- On the right side of the screen, various information about the meeting is displayed based on the action selected by the moderator on the control panel. For example, the results of the votes can be displayed here.

During the meeting, the web application also maintains a WebSocket-based connection with the server following the STOMP protocol [6]. In this way, the moderator can track how many people are currently present in the meeting from the entire membership. After starting the vote, it also becomes visible to him how many people have cast their votes up to that moment. Based on this real-time information, the moderator can start or close votes. The application also indicates with an appropriate error message if the WebSocket connection between the client and the server is broken. In this case,



(a) Meeting page

(b) Voting on mobile

Fig. 3: Web and Mobile User Interface

the client tries to reconnect every second and if successful, notifies the user that the connection has been successfully reestablished.

The application provides two different views for displaying the voting results. The user can list each participant's vote individually, or they can view the aggregate results in a pie chart.

C. Voting on Mobile

The primary purpose of the mobile application is to provide a platform for conducting votes. This section aims to present the main functionalities, architecture, and technologies used during the development of the mobile application.

Voting on a topic is the only functionality of the mobile application that is exclusively available to members. If a moderator attempts to log in, the system responds with an appropriate error message.

After a member joins a meeting, the mobile application retrieves the current meeting state from the server via the REST API and connects to the dedicated WebSocket channel to receive real-time updates, following the protocol described in V-B paragraph.

A meeting can have four states:

1) Scheduled Meeting - The moderator has not started the meeting yet.
2) Started Meeting without a topic - The meeting has started, but no topic is set for voting.
3) Started Meeting with a topic - The meeting has started with a topic available for voting.
4) Completed Meeting - The meeting has ended.

The mobile application indicates these states by displaying the corresponding message to users. Figure 3(b) shows a voting scenario for a topic.

The voting card displayed contains the topic's title, description, a list of options, and a submit button to confirm the vote. By default, no option is selected and the submit button is disabled. Users can select only one option from the list. Once a choice is made, the submit button becomes active, allowing the user to cast their vote. Upon pressing the submit button, the mobile client sends the vote to the server via the REST API and notifies the moderator through WebSocket that the user has voted. After submission, the button becomes inactive again, and a confirmation message appears below the voting card. A user can vote only once per topic.

VII. Deployment

The VoteHub application uses Docker containerization for both the server and web applications [7]. This approach allows developers to configure the runtime environment, relieving the host administrator from needing to know the system dependencies. As part of the deployment process in the GitLab CI/CD [8], the resulting Docker image is accessible from the GitLab Container Registry. A Docker image is a standardized package of the application that contains all the necessary files, binaries, and configurations to run the project. To build a Docker image, the project must include a Dockerfile, which contains the commands required to create the image.

Docker Compose is used to manage the created images efficiently. It simplifies managing services within the application, streamlines communication, and facilitates setting up network and data access connections. Configuration is done via a YAML file, which defines the services within the project. Docker Compose enables creating, starting, and stopping services.

The configuration responsible for deployment resides in a separate Git repository. This deploy repository handles set-

ting environment variables, accessing the server, and creating containers from the Server and Web application images. The Docker Compose YAML file manages five services:

1) Server
2) Web Application
3) MySQL Database
4) Speaches AI for transcript generation
5) MinIO file-based storage system

The deploy repository's pipeline provides three manually executable jobs: creating and running services, viewing log history, stopping and removing services.

VIII. Conclusion and future work

VoteHub is a highly customizable system that allows institutions to conduct their voting processes securely and efficiently.

The application can be extended with a number of other functionalities in order to be widely used. The following development options are planned:

- Identifying speakers during the automated transcript generation.
- Possibility for moderators to attach documents to the meeting topics, which will be available to the members.
- Possibility for creating subgroups within a forum for professional committees, and requesting/receiving preliminary opinion for specific topics from these committees
- Creating open meetings that can be followed by people outside the forum. In such meetings only forum members can vote, but outsiders can also follow the voting process. This could be useful for journalists, for example, to access meetings they might want to write about.
- Notifications in the mobile app for forum members about activities taking place within the forum.
- Introducing a chat interface within a forum.

References

[1] Singh, Pankaj, and Vikas Gupta. "JWT BASED AUTHENTICATION."

[2] Monday, Paul B. "Implementing the data transfer object pattern." Web Services Patterns: Java™ Platform Edition. Berkeley, CA: Apress, 2003. 279-295.

[3] Pimentel, Victoria, and Bradford G. Nickerson. "Communicating and displaying real-time data with websocket." IEEE Internet Computing 16.4 (2012): 45-53.

[4] Renzel, Klaus, and Wolfgang Keller. "Three layer architecture." Software Architectures and Design Patterns in Business Applications 10 (1997).

[5] Siriwardena, Prabath. "Openid connect (OIDC)." Advanced API Security: OAuth 2.0 and Beyond. Berkeley, CA: Apress, 2019. 129-155.

[6] Wang, Vanessa, Frank Salim, and Peter Moskovits. "Using messaging over Websocket with stomp." The Definitive Guide to HTML5 WebSocket. Berkeley, CA: Apress, 2013. 85-108.

[7] Miell, Ian, and Aidan Sayers. Docker in practice. Simon and Schuster, 2019.

[8] Cowell, Christopher, Nicholas Lotz, and Chris Timberlake. Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples. Packt Publishing Ltd, 2023.