AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Computer Science, Electronics and Telecommunications
Institute of Computer Science

FINAL PROJECT

POLLING SYSTEM WITH MEETING SCHEDULING SUPPORT

SYSTEM DO TWORZENIA ANKIET WSPOMAGAJĄCYCH ORGANIZACJĘ SPOTKAŃ

RADOSŁAW KOPEĆ, ALEKSANDRA MAZUR,
MATEUSZ SURJAK, MARCIN ZIELONKA

FIELD OF STUDY:
Computer Science

SUPERVISOR:
dr inż. Kamil Piętak

Kraków 2021

Contents

1. Project goals and vision

1.1. Characteristics of the problem

Many people participate daily in all kinds of meetings, both business and private. The organization of such an undertaking, especially in the current times of the coronavirus pandemic, may turn out to be a considerable challenge due to many details that need to be taken into account. Establishing the time, place of the meeting, and other key issues, which would correspond to each of the participants, repeatedly involves the enormity of the sent messages and the use of multiple platforms, and thus unnecessary confusion and stress. For this reason, applications allowing for the integration of these aspects are significant facilitation for organizers and participants.

A meeting is synonymous with the gathering of at least two people at a certain place and time. It seems that the biggest problem in planning such an undertaking is setting the appropriate date. To find the most adequate term, each person should provide their time availability. However, listing the individual hours and then selecting the most popular option is not a convenient method. This requires checking many proposals on the calendar and can lead to confusion. The solution to this problem may be to provide a more flexible choice of preferences, involving the continuous marking of areas of availability at intervals specified by the organizer. Not only more convenience would be provided, but also clarity.

Choosing a venue is another difficulty that must be faced during planning the meeting. Searching every available location is associated with additional dedicated time. Moreover, a comparison of all the proposals and selecting the most suitable seems to be burdensome, especially if we do not know the location of a given place. The solution to this problem may be to put all available proposals on the map so that everyone has the opportunity to make a choice in a simple and clear way.

Another possible obstacle is establishing additional details, e.g. assigning declarations to people, determining the dress code of the meeting or selecting the menu. Discussion on the aforementioned aspects is often carried out through separate communication channels, which significantly reduces efficiency and creates unnecessary confusion. Therefore, the solution to this problem is to provide people with the availability to create all kinds of surveys and declarations, so that all the arrangements for the meeting were gathered in one place.

1.2. Product vision

The aim of the project is to create a distinctive web application that combines many aspects that facilitate the organization of online or live meetings and gathers them in one place. Various additional modules will be provided to learn about participants' preferences. The most important of them will be the ability to choose the meeting date, enabling flexible selection of availability areas in given time intervals. The module for creating surveys with closed and open questions will be equally important. Application users will be able to create single-choice and multiple-choice questions and set the form duration. Additional functionalities will be the declaration assignment module allowing for the division of responsibilities between participants and the

place arrangement module, which will enrich the available locations with visualization on the map. All these aspects will be provided by the application called ProScheduler.

1.3. Project motivation

After carrying out deep research of available solutions, we discovered some applications offering functionalities we want to introduce as well. Despite the variety of applications for scheduling meetings, there is no ideal solution, since most of them focus only on one aspect of meeting scheduling, which is time. Unfortunately, we have not come upon any solution that offers something more. It is really dilemmatic from the perspective of people that want to go into details of the meeting or plan a business trip. For example, if we want to determine the time and carry out a survey among participants, we are forced to use Doodle.com and Google Forms, which necessitates the use of two different solutions instead of one.

In our product, we intend to close the whole flow of meeting scheduling in one application, introducing all crucial solutions for scheduling and making them intuitive and simple to use. We are convinced that such a solution is doomed to success because people like making life easier and using self-explanatory solutions that will save their time and ensure greater convenience. A good example is the well-known Uber, where instead of searching for a taxi company number and calling for a taxi, you can do it with only one click. You do not even have to worry about the payment because the application will withdraw money from the card itself. The simplicity, intuitive user interface, and orientation towards doing one and only one thing well are what guarantees success. Currently, meeting schedulers do not cover all needed features and forget about other needs such as meeting place choice, a survey came out, and declaration submission. Therefore, the available solutions will be presented and discussed below.

1.3.1. Available solutions

1.3.1.1. Doodle

Doodle [13] is a web application that enables users the functionality to organize meetings with a voting system for the suitable time slots entered by the organizer. However, in a situation where we want to have many, densely arranged slots, the discrete-time intervals used in Doodle are often inconvenient, as we need to create each of them separately. For continuous-time intervals, this problem does not occur because we can select many slots in one simple step with greater accuracy.

The additional feature is a virtual calendar in which all actual user meetings are displayed. Currently, the platform does not support the possibility of choosing a meeting place, carrying out surveys, or asking questions.

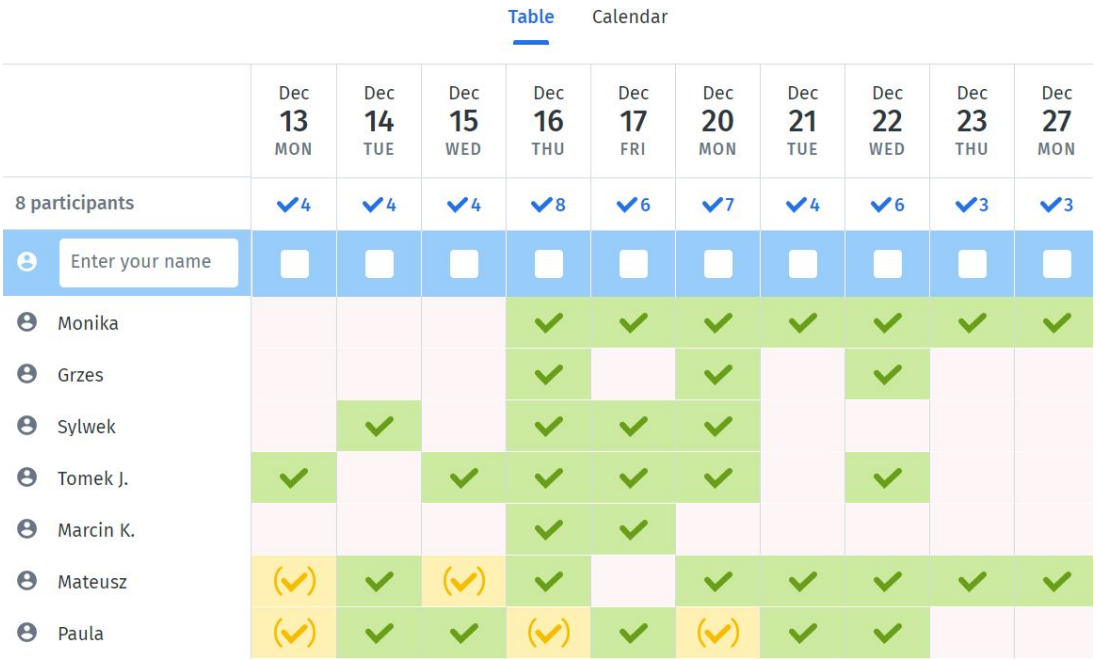

Figure 1: Voting results view

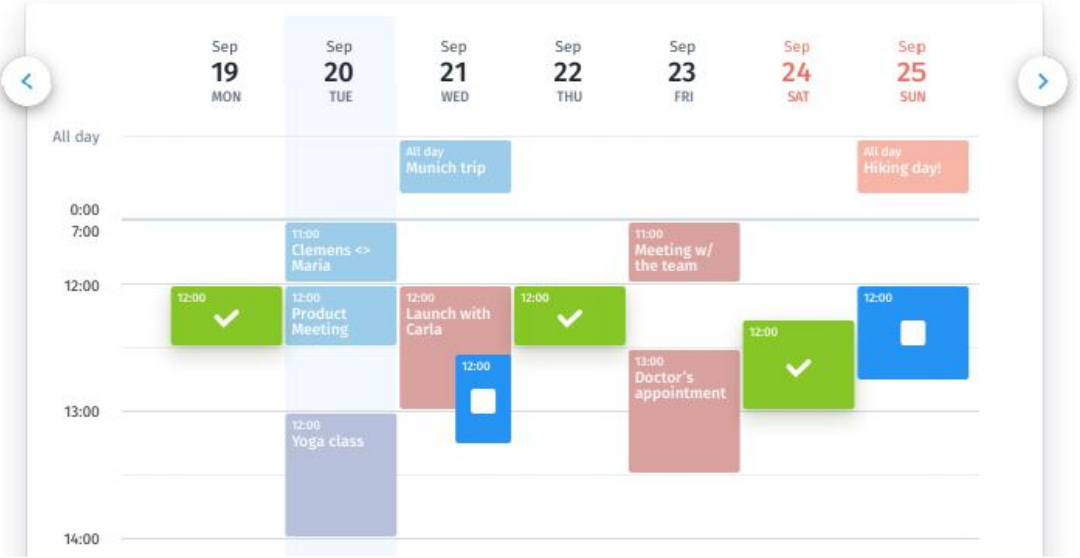

Figure 2: Virtual calendar view

1.3.1.2. StrawPoll

StrawPoll [55] is a web platform that enables users to create generic and dedicated time surveys. Live voting results preview is what distinguishes this application from others. Additionally, organizers can set a deadline to fill up the survey. It is worth mentioning that there are two types of surveys, private and public. The first of them can be viewed by specific invited users, while the second by anyone with the link. Users can modify the answers all the time before the deadline and can vote anonymously. However, this platform does not have a dedicated solution for selecting a meeting place and integrating all meeting details in one specific panel.

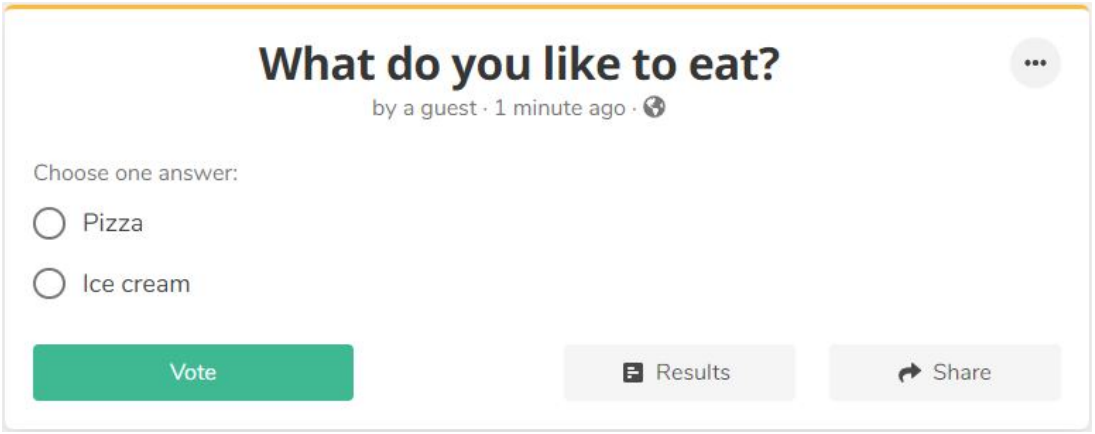

Figure 3: Question view

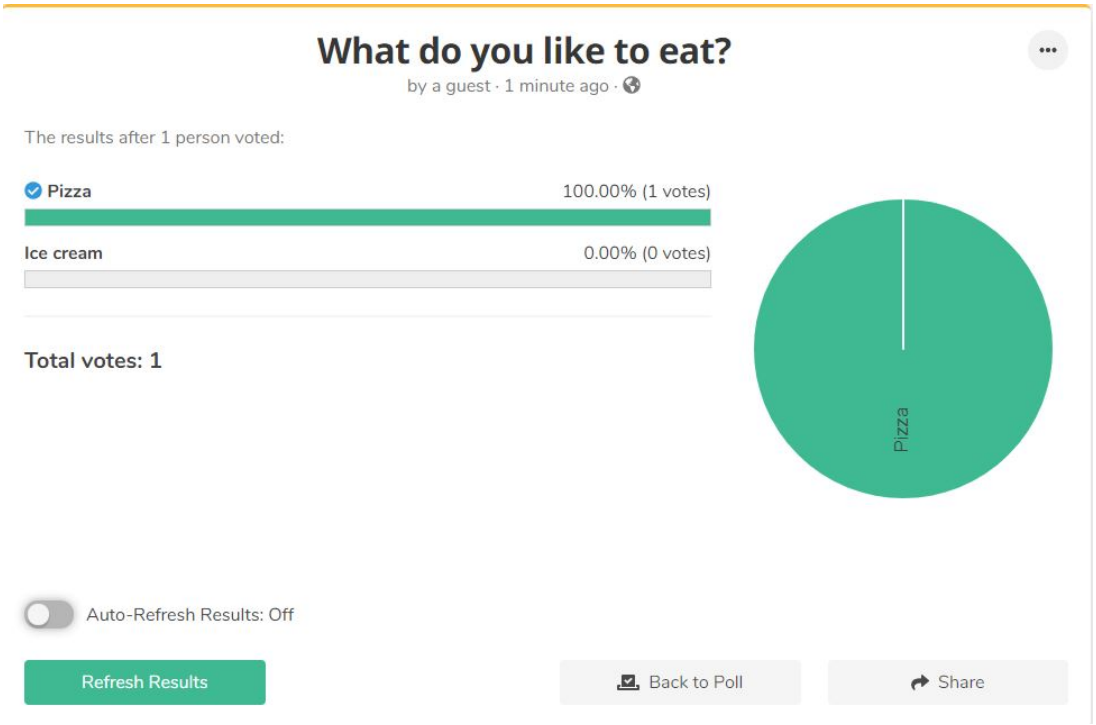

Figure 4: Survey results view

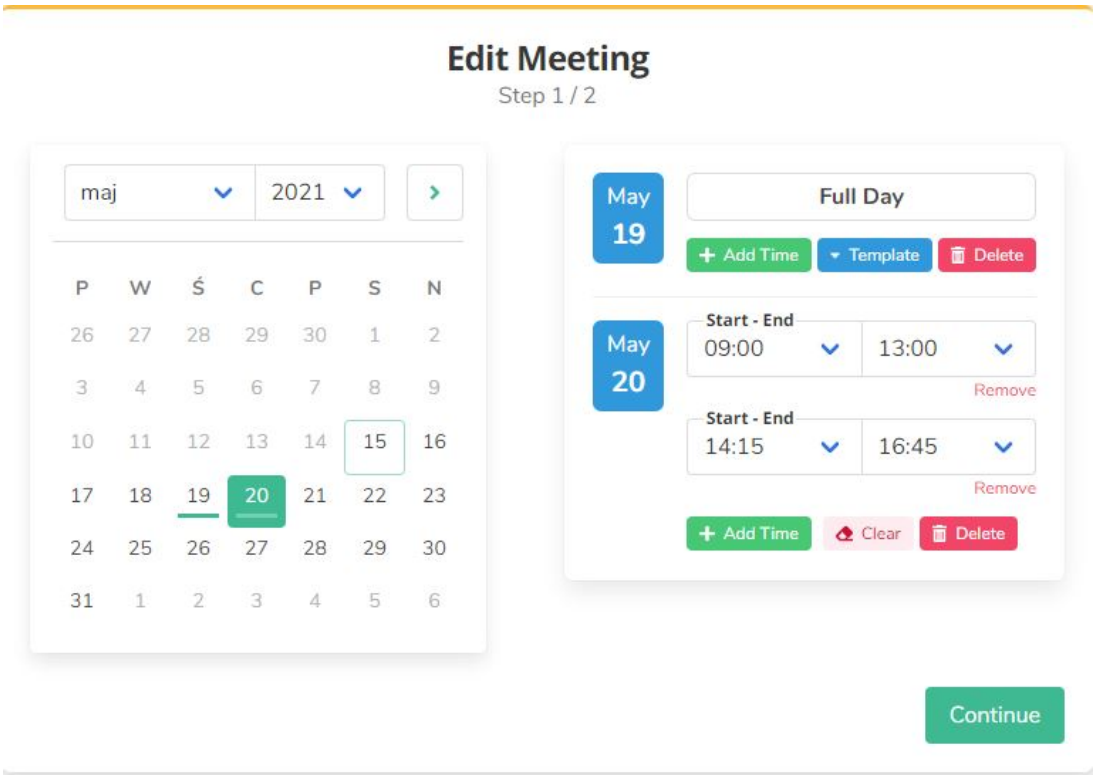

Figure 5: Creating a meeting

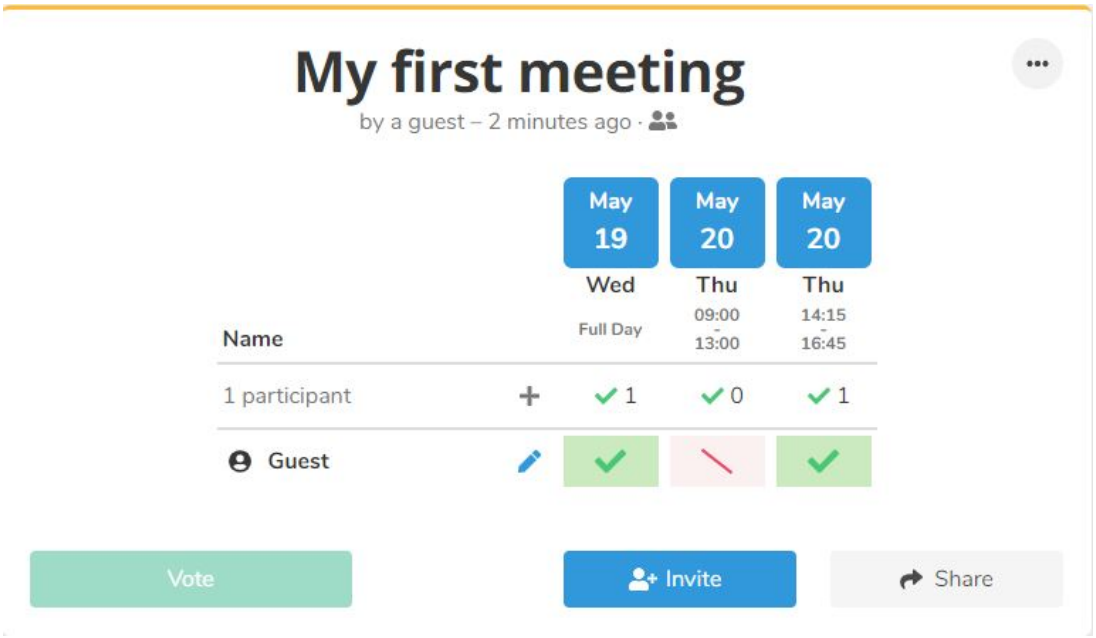

Figure 6: Meeting time results view

1.3.1.3. Google Forms

Google Forms [22] is a solution provided by Google, which allows creating extended surveys. Using this platform users can create a survey that contains:

- open questions
- single choice questions
- yes or no questions
- linear scale questions
- multi choice questions

This solution is only oriented towards creating surveys and it does not support any additional functionalities. It does the one thing well and because of that we will implement the majority of Google Forms features in our application by supporting different types of questions.

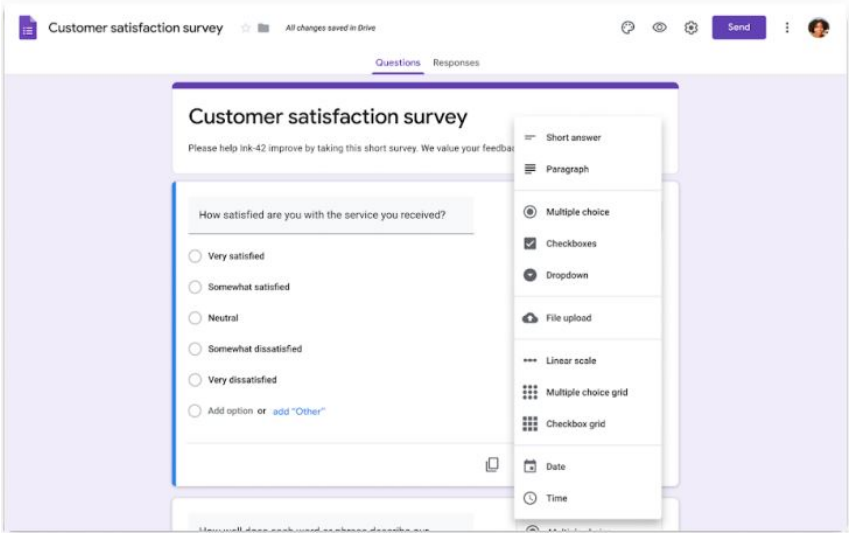

Figure 7: Questions view

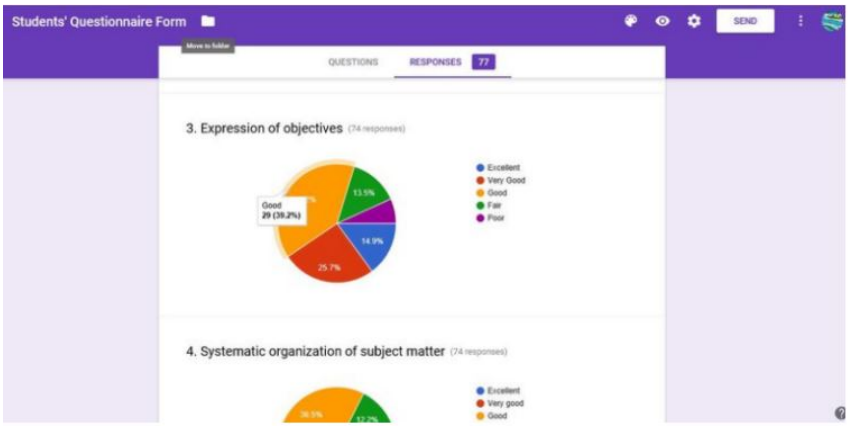

Figure 8: Survey results view

1.3.1.4. WhenAvailable

WhenAvailable [59] is a commercial solution that imposes limits on a free version. This solution offers the possibility to choose a meeting time and place. Unfortunately, the creation of a meeting is very time-consuming because by selecting one day, the organizer has to manually input all time slots. There is not a drag & drop option and the meeting place is high-handedly enforced by the organizer, so the users cannot vote and select a different one.

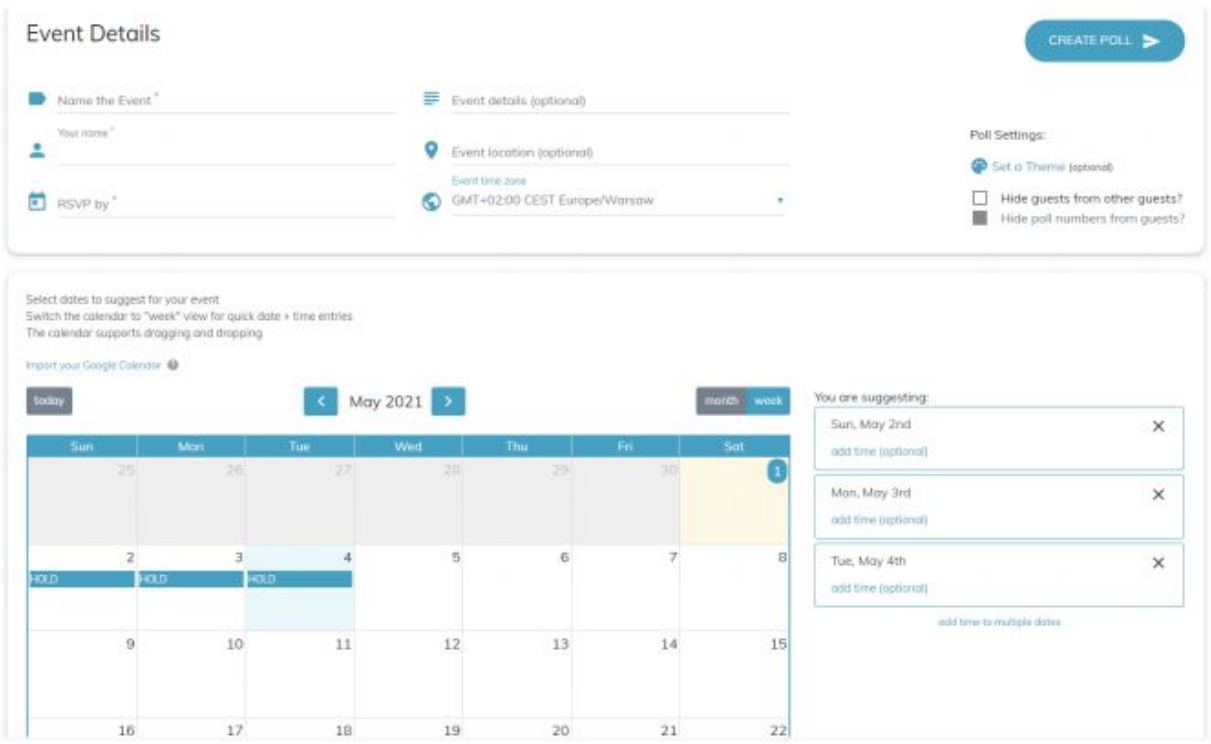

Figure 9: Scheduling a meeting

To sum up, there is a range of solutions that can be used to plan a meeting, but unfortunately, there is not any platform that links all the needed features. That is why in our application, we want to meet the needs of users and create a single platform that will cover all the details related to meeting planning, and we hope it will eliminate the necessity of using multiple solutions.

1.4. Threat analysis

System components with individual stages of their implementation that may cause difficulties or have a negative impact on the successful delivery of the product are presented below.

1.4.1. Demanding user interface

The final user interface representing all functionalities of the product will include a large number of different components which are expected to be highly integrated with themselves. What's more there are plans to provide integration with other third party services like Google Calendar, for instance. As a result, such a system complexity from both business and technical side may turn out to be laborious. The number of user views planned to be realized is significant and implementation of some components could be time-consuming. One way to solve this problem will be to assign a sufficiently high priority to tasks associated with the user interface, as well as to use external libraries offering ready-made individual components.

During the product development, the user interface actually turned out to be challenging. Therefore, in order to successfully complete all related tasks and achieve the intended goals, we have created mockups of all expected user interface components first and then have started their implementation. Such a way to deal with the difficulty allowed us to separate functional and technical related challenges. As a result, we have successfully achieved all the designated goals in time. What's more, the expected difficulties have not disturbed the development process on any of its stages

1.4.2. Demanding user experience

Ensuring a positive user experience while using the product is essential. The application should be designed so that the end-user can use it easily and intuitively. To learn the opinions of users and introduce suggested changes, UX research will be conducted at several stages of the project development.

In order to achieve as intuitive and simple user interface as possible, we designed and created initial mockups presenting the main idea of how the application would look like. Only then did we start implementing the product. Such an approach made it significantly easier to design and implement a user-friendly interface. However, despite applying such a way to develop a product, we have not avoided all the difficulties. We have come to a conclusion that our early design is not intuitive enough after implementing the first components and views. Therefore we have decided to redesign existing mockups and have proceeded to refactor the frontend web application. As a last resort, all these actions make the final product simple and intuitive - in accordance with the customer's initial intentions and requirements. The achieved goals are confirmed among others by the carried out UX tests in the final stage of product development.

1.4.3. System security

Due to the planned system architecture based on microservices, as well as the use of external authentication systems, designing appropriate security can be complicated. The management of sensitive data is very important, so proper implementation will have to be carefully thought out.

One of the ways to deal with this issue will be, as in the previous cases, assigning a sufficiently high priority to tasks related to it, as well as following current trends and solutions in other systems of this type operating on the market.

In order to eliminate any plausible problems related to system security like users' accounts, their authentication or authorization - we focused on it from the early beginning of the product development. We decided to use third party services like Google and Facebook for authentication to avoid storing sensitive data like user password in the database. What's more we have designed a custom internal library for handling authorization in the same way in all of the microservices which make up the whole system. Therefore, we have succeeded in focusing on other aspects of the project in further stages of product development.

2. Functional scope

2.1. Context of using the product

2.1.1. Characteristics of the system users

The designed solution is dedicated to:

- any person who is looking for an intuitive and simple solution to organize a meeting (whether private or business)
- medium and smaller teams that need to determine the details of a given meeting, conference, etc., where the current solutions available on the market are insufficient or force them to use multiple communication channels at the same time.

Due to the nature of the designed system and its functionality, two actors can be distinguished - the meeting organizer and the meeting participant.

A meeting participant is a person who has joined an existing meeting in the system by himself or has been invited by the meeting organizer. Such a person has access to information about a given meeting/event, such as name, description, other participants, etc., and may participate in active polls created by the meeting organizer or in other voting-related to the time or place of the event.

The organizer of the meeting is the person who has created a new meeting in the system. This person can fully configure the created meeting by setting possible time frames, meeting places, or inviting and removing participants. In addition, the organizer has the right to create dedicated surveys and send notifications to other participants. All these privileges may also be given to a person appointed to this position by another meeting organizer.

2.1.2. Characteristics of external cooperating systems

In order to create an intuitive system associated with popular and widely used other solutions on the market - the target system will be integrated with the following external systems:

- Google Accounts [20]
- Facebook [15]
- Google Calendar [21]
- OpenStreetMap [44]
- Email services

2.1.2.1. Google Accounts

The target product will be integrated with Google Accounts in order to provide the ability for the user to log in via their Google account. This will save resources needed to implement our own authentication system and create other functionalities of key importance for the system. Moreover, it will make it easier for users to create an account and eliminate the need to remember another password.

2.1.2.2. Facebook

The target product will also be integrated with the Facebook platform on the same terms as Google Accounts - in order to be able to log in users via their Facebook account. This will provide users with an additional login method, as a result of which they will not have to use one predefined method, but will be able to choose the one that satisfies them more.

2.1.2.3. Google Calendar

Integration of the system with Google Calendar services will consist in the possibility of saving meetings to which it is assigned in the user's personal calendar. This will allow meeting participants to use the delivered product more conveniently and to organize their time more efficiently.

2.1.2.4. OpenStreetMap

Product integration with OpenStreetMap services will facilitate the possibility of selecting the meeting place by the organizer and participants by marking it on the map (instead of entering the exact address). This will allow users to enhance the experience while using the system and will reduce the additional time spent on searching for a given place.

2.1.2.5. Email services

The implemented product will also offer the possibility of sending notifications to meeting participants via email. This will allow the meeting organizer to better communicate with other participants, among others, by sending all kinds of reminders.

2.2. Detailed description of the functions

The main aim of the application is to facilitate the organization of meetings by a flexible selection of time frames and creating surveys.

2.2.1. Registration and login

As mentioned before, registration and login will be provided through the use of Facebook and Google, which will ensure the reliability and security of sensitive data.

2.2.2. Creating a meeting

The user will be able to create any number of meetings, providing a name and description and invite participants using their email addresses. The functionalities provided by the system can be divided into several areas due to their nature related to the organization of the meeting:

- meeting time
- place of a meeting

- surveys
- notifications of participants
- declarations

2.2.3. Area: meeting time

This area includes all the functionalities for determining the time of the organized meeting between the organizer and all participants.

Examples of functional requirements included in this area:

- possibility to create a meeting with a time selection
- possibility to mark possible dates of the meeting
- possibility of marking time availability in a given meeting
- setting the final date and time of the meeting

2.2.4. Area: place of a meeting

This area includes all the functionalities related to the arrangement of the meeting place between the organizer and all participants.

Examples of functional requirements included in this area:

- possibility to create a meeting with a choice of place
- possibility to mark proposed meeting places on the map
- the ability to display on the map possible meeting points marked with pins

2.2.5. Area: surveys

This area includes all the functionalities for creating and managing surveys available during a given meeting, including creating surveys by the organizer, filling them in by participants, and previewing the survey results.

Examples of functional requirements included in this area:

- the ability to create a survey with open and/or closed questions
- the ability to display the results of the survey after its completion
- the possibility of entering answers in all kinds of questions in the provided survey

2.2.6. Area: notifications of participants

This area includes all functionalities regarding notifications between participants and the organizer, including sending reminders about the meeting, filling in the required information, etc.

Examples of functional requirements included in this area:

- possibility of receiving a notification and reminder about an available survey/meeting or a new invitation
- the ability to cancel a previously scheduled meeting and then send a notification to each participant
- possibility of sending a custom notification to each participant

2.2.7. Area: declarations

This area includes all functionalities related to the participant declaration system, including submitting various types of commitments within a given meeting, such as taking a map, first aid kit, etc.

Examples of functional requirements included in this area:

- the ability to add a declaration regarding for example the assignment of people responsible for transport or the required purchases and the ability to send notifications to participants so that they can assign themselves to the declaration
- the possibility of assigning yourself to a given declaration

2.3. User stories

After careful analysis and reflection, user stories have been grouped according to their priority in the order of their realization and implementation. On this basis, the following four categories have been created.

- **MUST** - key requirements that must be compulsorily fulfilled
- **SHOULD** - requirements that should be met in order for the delivered solution to compete with the available alternatives
- **COULD** - requirements that extend existing solutions, however, are not essential for the proper operation of the system
- **WOULD** - requirements to consider for implementation that enhance product quality and user experience

2.3.1. MUST priority functionalities

2.3.1.1. Organizer

- As an organizer, I want to sign up for the application using third-party auth providers, to not create an additional account.

- As an organizer, I want to sign in to the application.
- As an organizer, I want to sign out of the application.
- As an organizer, I want to create a new meeting with a name and description, so I can better distinguish meetings.
- As an organizer, I want to add participants to the meeting, so I can share it with others.
- As an organizer, I want to remove participants from the meeting, to prevent unwanted people from attending.
- As an organizer, I want to create a meeting by setting up a time survey, so that participants could mark their time availability.
- As an organizer, I want to set up a time survey which time-spans are possible to mark to determine that at this time the meeting can be set up.
- As an organizer, I want to set up a time survey which time-spans are impossible to mark to determine that at this time the meeting cannot be set up.
- As an organizer, I want to create a survey attached to the meeting to find out more about the participants and their preferences.
- As an organizer, I want to add to the survey open and closed questions, to better define the form of the answer.
- As an organizer, I want to set a deadline for filling up the survey, to mobilize participants to answer.
- As an organizer, I want to share a survey to participants attached to the meeting so that they could fill it up.
- As an organizer, I want to send notifications about the upcoming meeting to all participants on the day of the meeting, so that no one will forget about it.
- As an organizer, I want to send notifications about a survey to participants at any time, to remind them to complete it.
- As an organizer, I want to get survey results after closing it to get to know the participant's answers.
- As an organizer, I want to see real-time results based on already filled-up surveys, to know the most frequently given answers in advance.
- As an organizer, I want to see how many and which participants have already filled up the survey, so that I can remind others to respond.
- As an organizer, I want to create an online meeting with the ability to attach a link to the site where the meeting will take place (e.g. on Webex, Zoom, Facebook, etc.) so that participants have this data in one place.

- As an organizer, I want to have a list of meetings organized by me to not forget any and manage them better.
- As an organizer, I want to edit meeting details after its creation.

2.3.1.2. Participant

- As a participant, I want to sign up for the application using third-party auth providers, to not create an additional account.
- As a participant, I want to sign in to the application.
- As a participant, I want to sign out of the application.
- As a participant, I want to mark my time availability at the meeting so that the organizer knows when I have time.
- As a participant, I want to choose one or more answers in closed questions in a survey to express my opinion as best as possible.
- As a participant, I want to write answers to open questions in a survey so that the organizer knows my answer.
- As a participant, I want to receive notifications of new/not filled surveys so that I do not forget about filling them.
- As a participant, I want to receive notifications of upcoming meetings so that I do not forget about them.
- As a participant, I want to have a list of meetings I attend to not forget any and manage them better.
- As a participant, I want to view the given meeting details.

2.3.2. SHOULD priority functionalities

2.3.2.1. Organizer

- As an organizer, I want to send to participants repeated notifications for some reason (e.g. they haven't filled up the survey yet) to get some action.
- As an organizer, I want to cancel the meeting at any time.
- As an organizer, I want to send a proper notification about cancellation to participants attached to the meeting so that they can schedule their time differently.
- As an organizer, I want to create a commitment attached to the meeting (e.g. "Who will take a map or spare batteries?") so that I could better divide tasks.
- As an organizer, I want to send notifications to participants about created commitments so that they can review them.

2.3.2.2. Participant

- As a participant, I want to quit the meeting at any time and notify other users in the meeting about this fact so that they know that I will not be there.
- As a participant, I want to be assigned to the meeting commitment so that I know if I should do or take something.
- As a participant, I want to change previous answers in an already filled-up survey to be able to select the answers that are more relevant at the moment.

2.3.3. COULD priority functionalities

2.3.3.1. Organizer

- As an organizer, I want to add to another participant in the meeting the role of organizer (so with all permissions granted to the meeting organizer) so that I can share responsibilities with others.
- As an organizer, I want to share a meeting with a generated link or code to allow others to join the meeting more easily.

2.3.3.2. Participant

- As a participant, I want to add the meeting to my Google Calendar to better plan my time.
- As a participant, I want to use my Google account profile avatar if an account was created using Google auth provider so that I do not have to additionally set the photo.
- As a participant, I want to use Facebook account profile avatar if an account was created using Facebook auth provider so that I do not have to additionally set the photo.
- As a participant, I want to edit my profile to update my data.

2.3.4. WOULD priority functionalities

2.3.4.1. Organizer

- As an organizer, I want to add the possibility via configuration to allow other participants to invite people.
- As an organizer, I want to create a meeting with the option to choose a possible place where the meeting takes place so that others can choose which one suits them best.
- As an organizer, I want to choose which places are available to vote on/choose by participants in the meeting to suggest possible seats.
- As an organizer, I want to decide if participants in the meeting can add their place to the meeting vote.

2.3.4.2. Participant

- As a participant, I want to see a map with all possible places for a meeting (e.g. pins on a map) to get a better idea of where these places are located.
- As a participant, I want to add a new place to the meeting vote if it is allowed by the organizer to propose a better option.
- As a participant, I want to see the history of joined meetings so that I can find the information I need.
- As a participant, I want to have a chat within a meeting to easily communicate with other participants.
- As a participant, I want to share some costs related to the meeting with other people and have a clear summary of who owes money (and how much) to facilitate reimbursement.
- As a participant, I want to integrate with my Google Calendar to see if the meeting collides with any saved in Google Calendar so as not to additionally check for collisions.

2.4. Non-functional requirements

2.4.1. Web application

The requirements listed below are related to web application, which is part of the solution.

2.4.1.1. Intuitive interface

The web application should be designed in the most intuitive way. The user should not have any problems or doubts in navigating the website and performing various types of actions. Additionally the application should scale well on mobile devices.

2.4.1.2. Application performance and reliability

The provided solution should be highly scalable and able to handle the middle load. The number of requests per second should be about 20. The scalability guarantees a higher amount of requests per second, but due to the limited resources (cost of application servers), the final solution will accept the middle load. Additionally, the system should be immune to outside attacks like CSRF, XSS, etc. There will be proper protection on the user interface side as well as on the API and server-side. The system architecture is designed with recommended guidance in order to minimize potential attack scenarios.

2.4.1.3. Application security

The web application should use the HTTPS protocol to encrypt the data flow. Additionally, the system will use remote authentication providers like Google Accounts in order to minimize sensitive data storage on our side.

2.4.1.4. Other application integration

The system should integrate with other remote applications as much as possible, to make the delivered solution very intuitive while using it to the end-user.

2.4.2. Business aspects

As part of non-functional requirements related to business aspects, the provided solution should be designed with a view to potential commercial market introduction at a later time. The application should be prepared for the greater system load associated with serving multiple users. Additionally, in order to expand the product, scalability should be provided.

2.4.3. Legal aspects

As part of non-functional requirements related to legal aspects, the provided solution should be designed in accordance with current functioning law, including the aspect of user personal data storage.

2.5. Description of the main use cases

2.5.1. Meeting time module

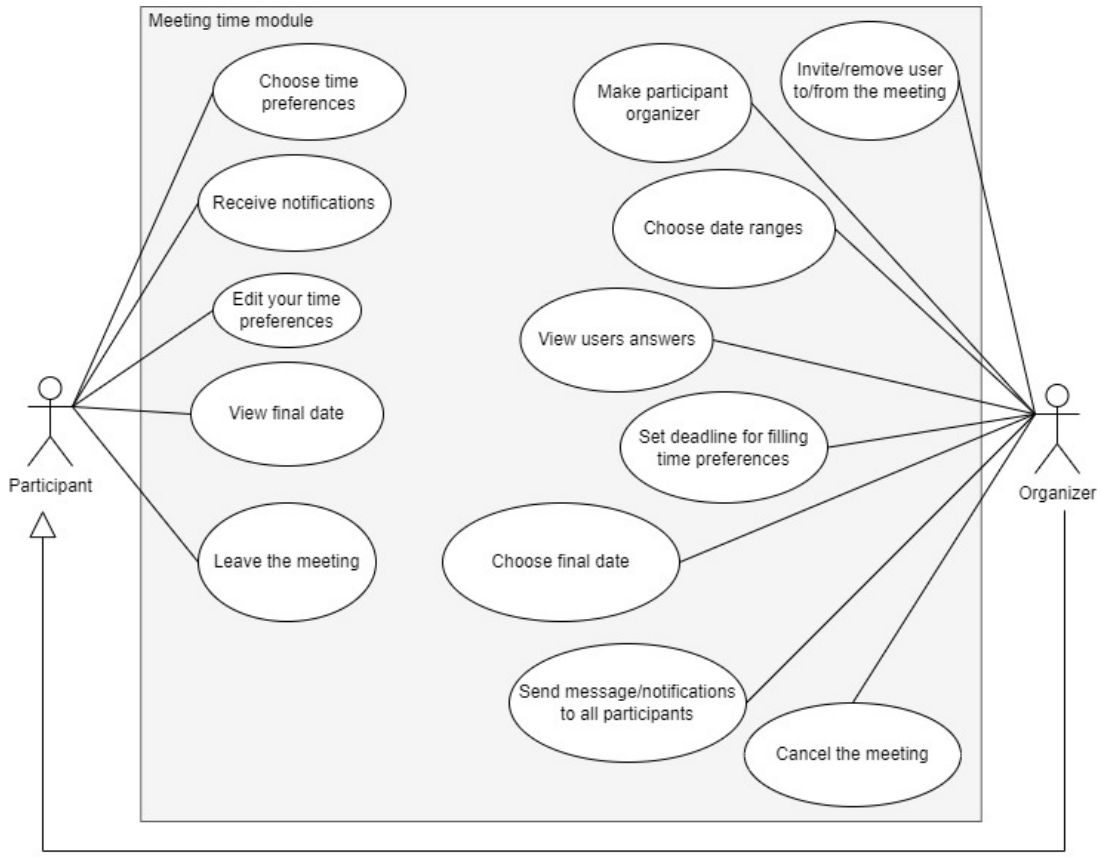

Figure 10: Meeting time module - use case diagram

The diagram in Figure 10 includes basic time module use cases as well as some notification features. The organizer chooses data ranges in which participants can fill their preferences or sets the final date instead. Participants can leave the meeting and send the message with absence justification to the organizer (optional). The organizer is able to make another participant an organizer as well as remove a participant from the meeting. Participants receive messages from organizers and notifications about the incoming meeting.

Happy Path:

1. Organizer creates a new meeting with data ranges.
2. Organizer invites people to the meeting.
3. User 1 accepts the invitation
4. User 2 accepts the invitation
5. Participant 1 fill their preferences about meeting time
6. Participant 2 fill their preferences about meeting time
7. Organizer confirms a final date
8. Participants receive notification about the final date
9. Participants receive notification 1 day before the meeting.

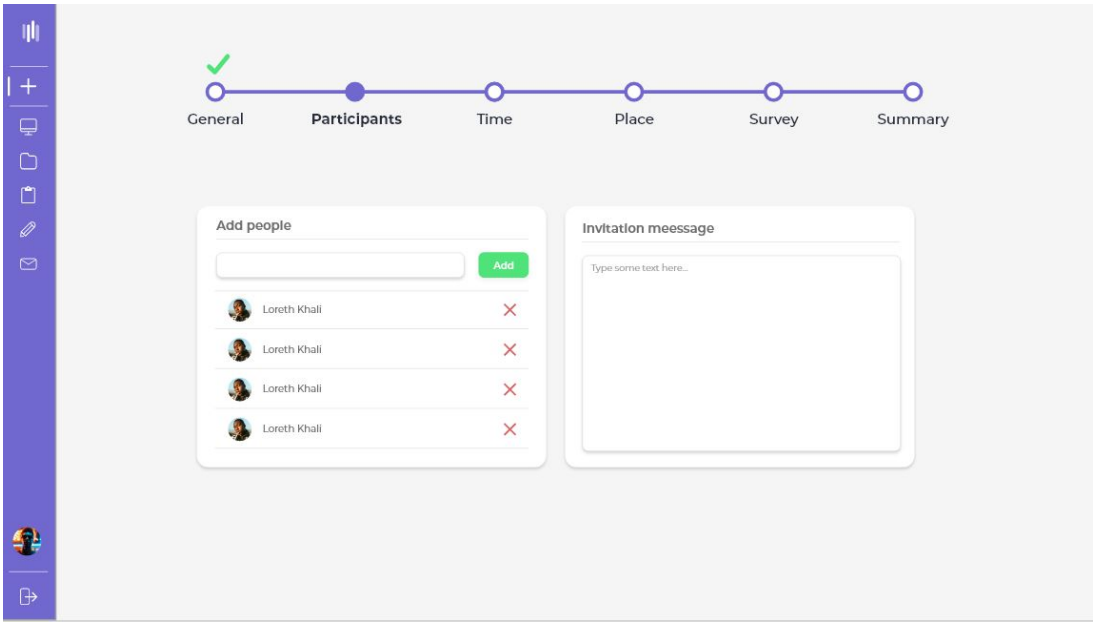

Figure 11: Inviting participants - mockup

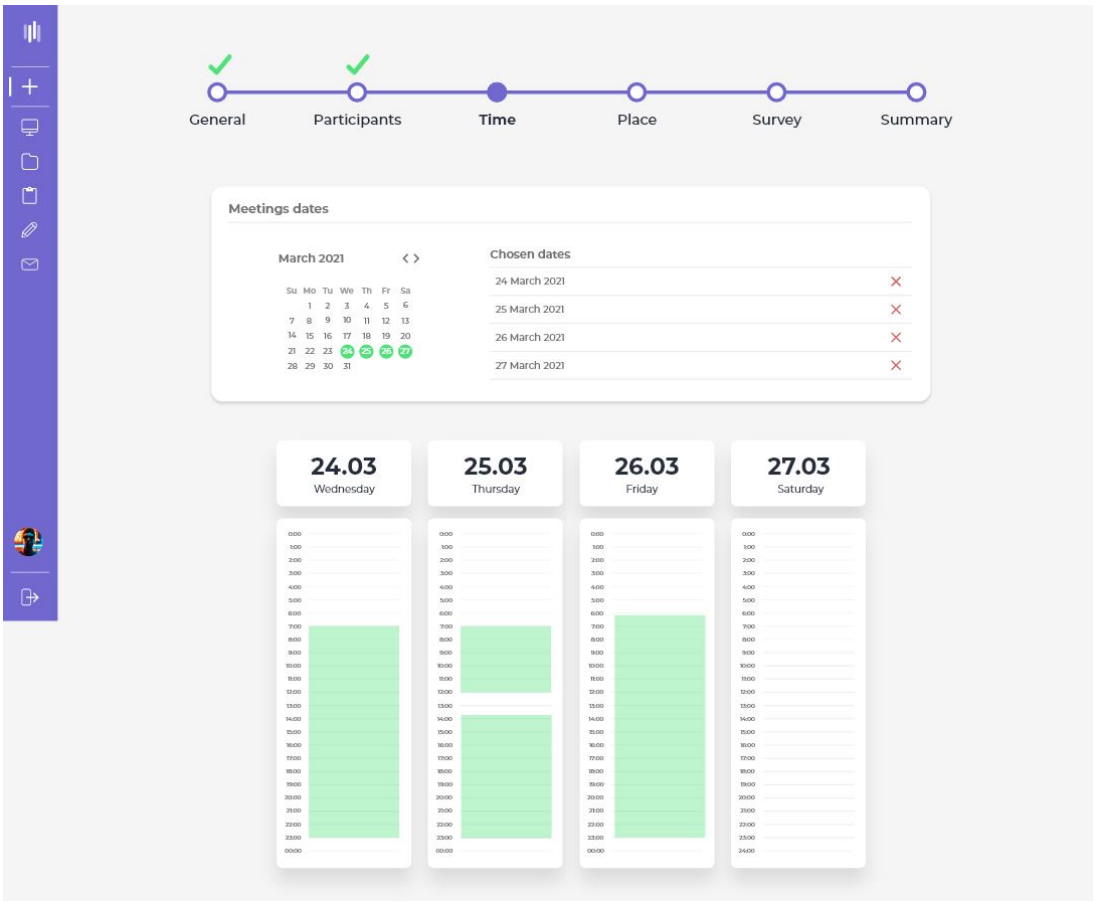

Figure 12: Choosing time interval - mockup

2.5.2. Meeting place module

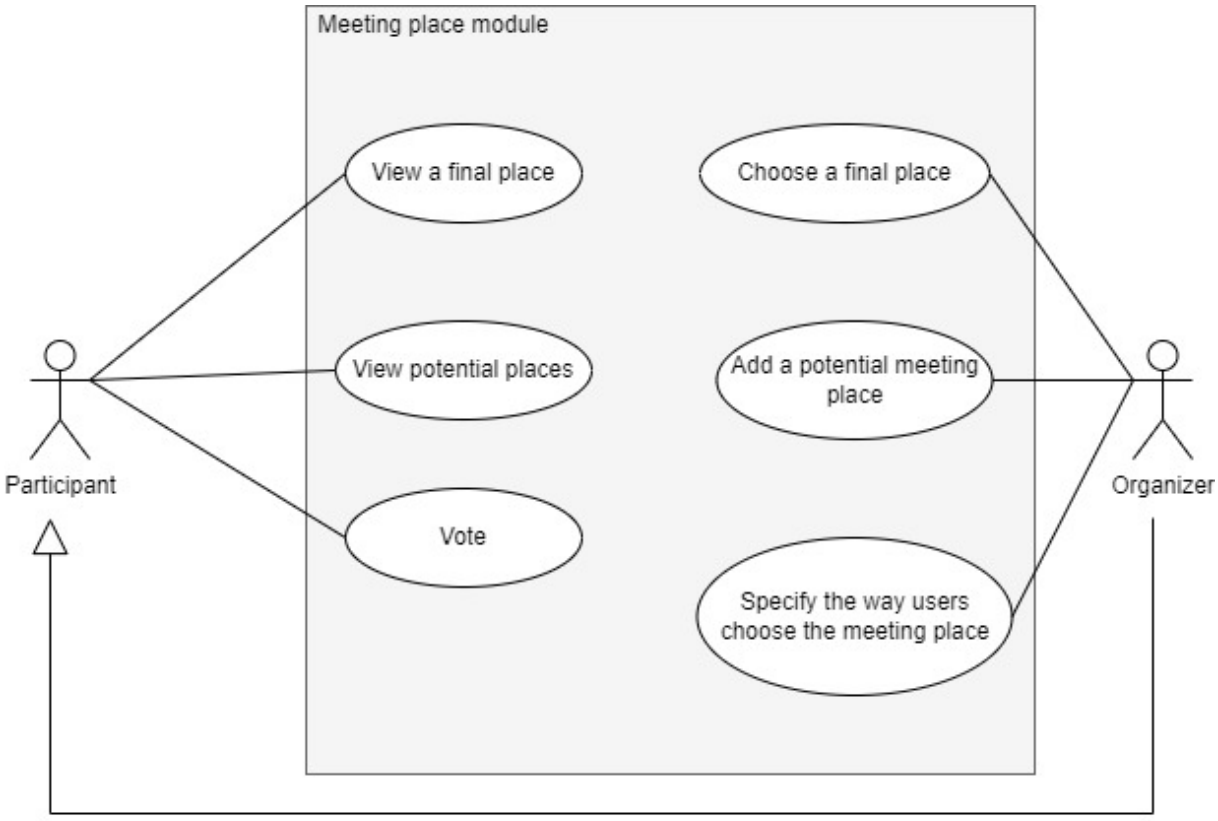

Figure 13: Meeting place module - use case diagram

The organizer adds a place module to the meeting and chooses whether the meeting should be online, or live as is shown in Figure 12. The next step is to indicate the proposed places on the map, but it is also possible to skip this stage and choose the final place. Participants can vote for a given place or add their proposals if the organizer has allowed such an activity. After voting is complete, users can see the final date. It is worth noting that the organizer is also a participant.

Happy path:

1. Organizer enables place module.
2. Organizer set Google Map as a way to choose the meeting.
3. Organizer puts some marks on the map.
4. Participant 1 looks for the map and votes for place 1.
5. Participant 2 puts their place.
6. Participant 1 changes the vote for place 3.
7. Organizer confirms a final place as place 3.
8. Participant 1 can view the final place.

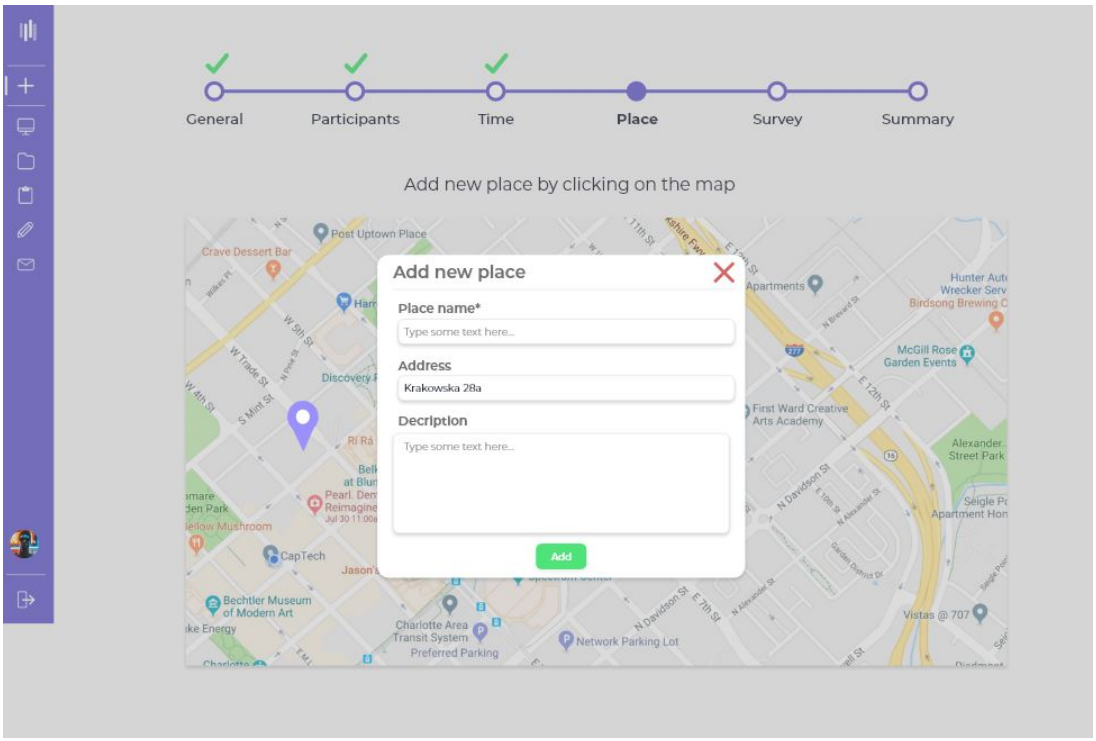

Figure 14: Adding new place on the map - mockup

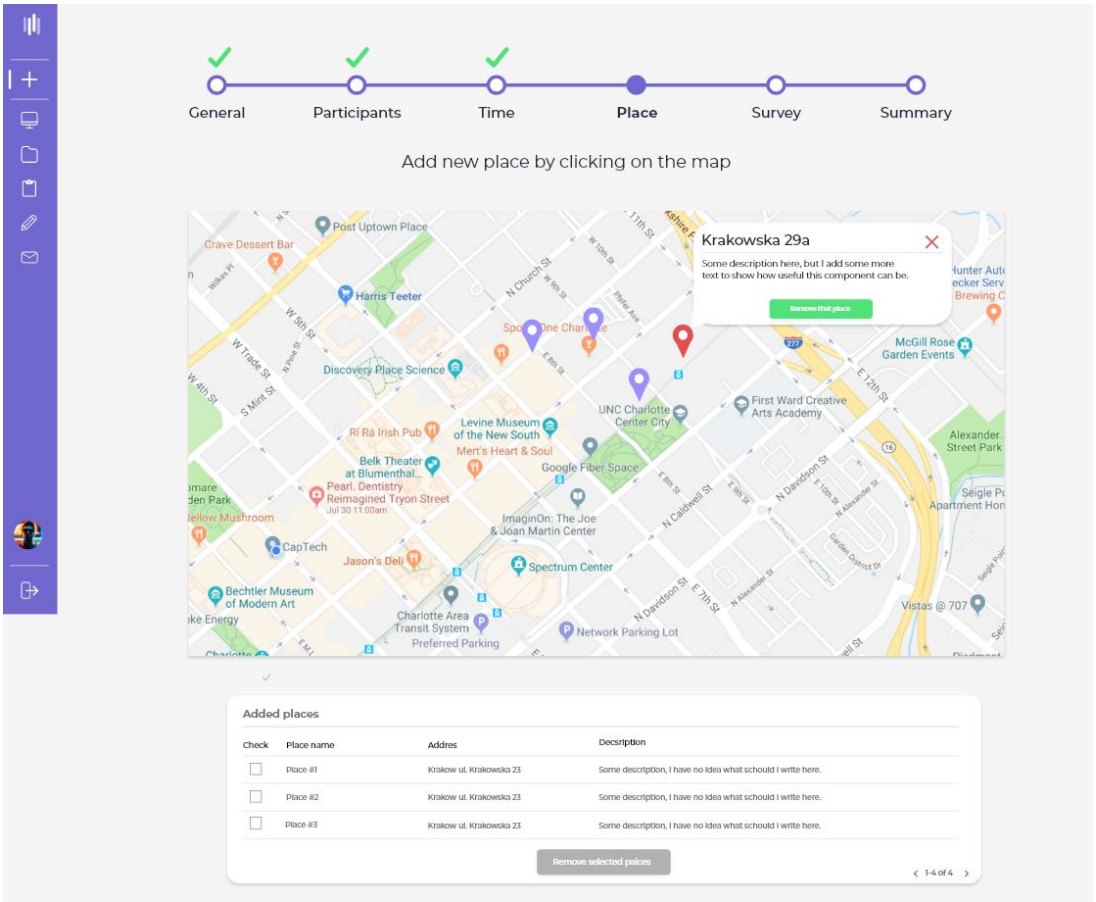

Figure 15: Displaying all places on the map - mockup

2.5.3. Survey module

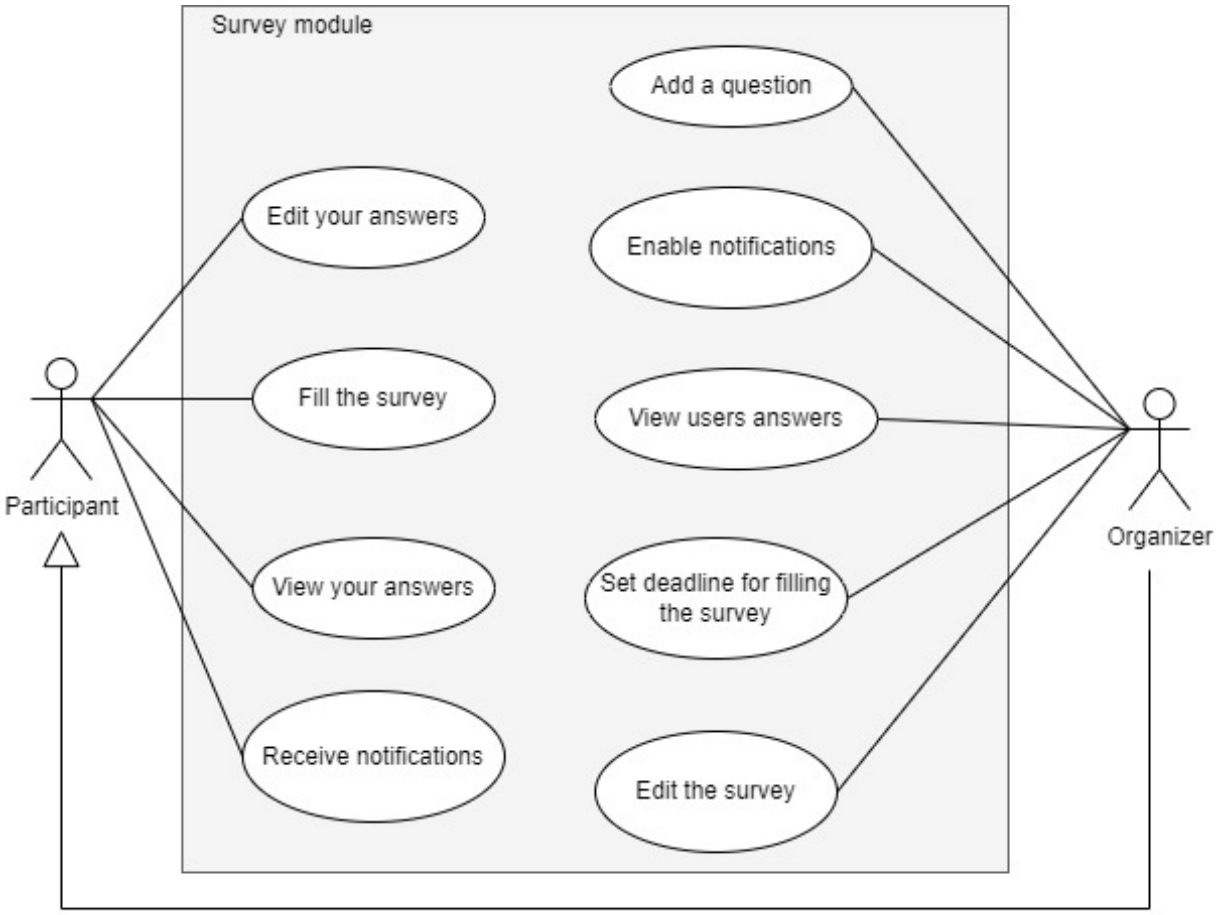

Figure 16: Survey module - use case diagram

Organizers can create surveys in the meeting and after that run a survey to allow participants to fill it as is shown in Figure 14. The organizer can enable the notification to inform participants about the survey or notify anyone who has not filled the survey yet. At any time if the survey runs, the organizer can view the answers.

Happy path:

1. Organizer creates the survey with sort of different questions.
2. Organizer enables notification
3. Participants receive notifications.
4. Participant 1 fills the survey.
5. Participant 2 fills the survey.
6. Organizer views the current answers.

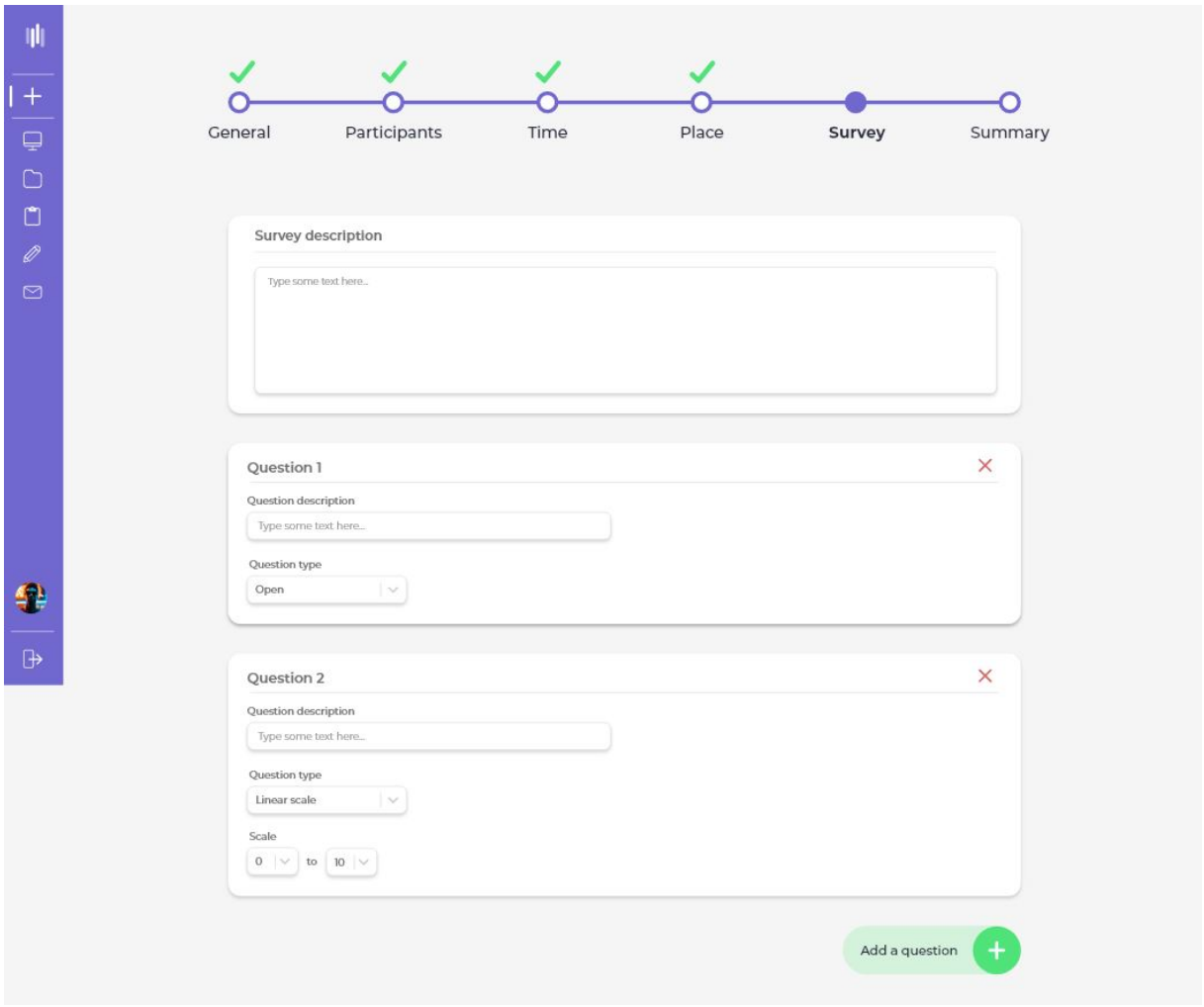

Figure 17: Creating a survey - mockup

2.5.4. Declaration module

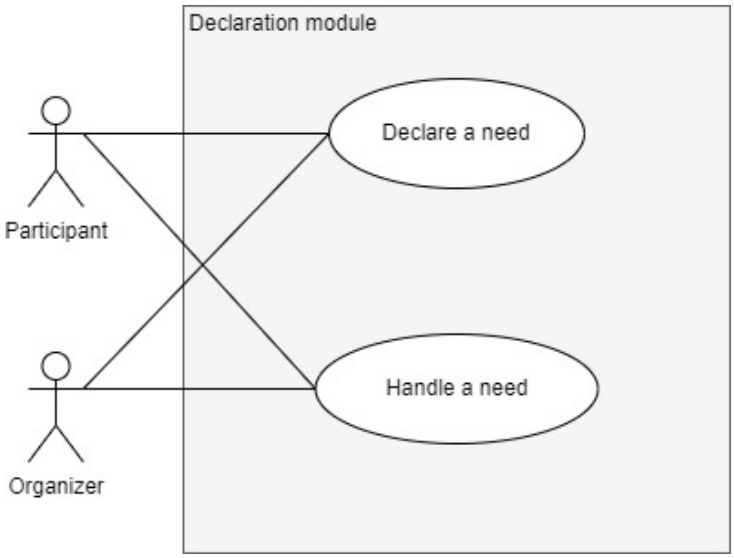

Figure 18: Declaration module - use case diagram

The organizer adds a declaration module to the meeting then participants can declare their own needs as is shown in Figure 16. Everyone can take responsibility for another's needs.

Happy path:

1. Organizer enables declarations for barbeque.
2. Organizer asks about participants' needs such as meat, grill coal, etc.
3. Participant 1 declares to take meat.
4. Participant 2 declares to take grill coal.

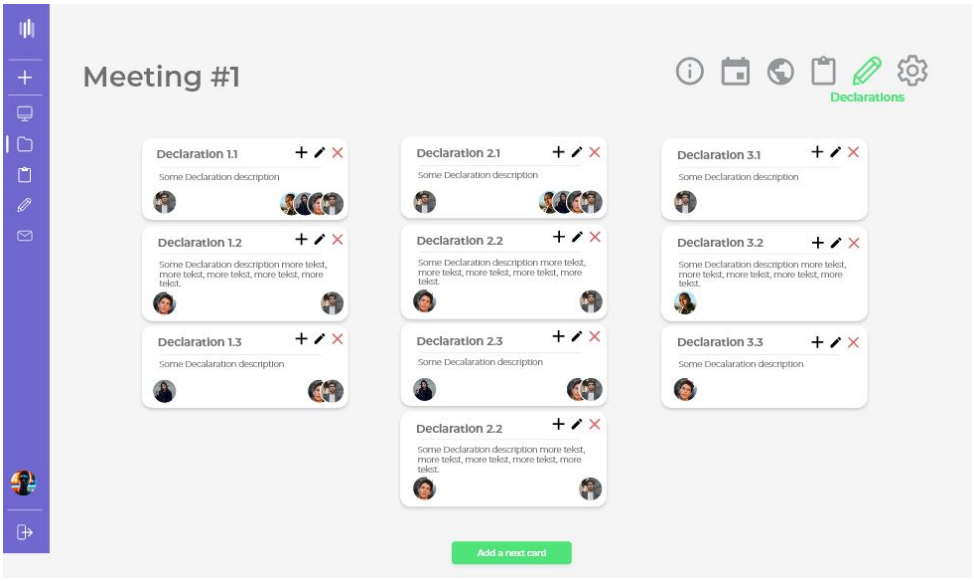

Figure 19: Displaying all declarations - mockup

3. Selected realization aspects

3.1. System architecture

Architecture is the most crucial part of every system, it should be well-considered and designed in an elastic way to pave the way for future extension and modification. For that reason, we paid particular attention to this aspect.

We faced a choice if our application should be a monolith or should be designed by following the current standards and trends by the convention of microservices. As we care about dealing with the system architecture theme in the most professional way and challenging ourselves in that matter, we decided to build the system based on microservices. We were aware that it can be a huge challenge because of the lack of prior experience with such architecture, but the pros and benefits that can be achieved due to microservices tipped the scale. With microservices, we create each service in the cloud completely independently from others, in different programming languages. We got out of unnecessary dependencies because services forced on us the separation of particular responsibilities. Our system has been designed in a way enabling continuous extension, growth, elasticity, and high availability.

We distinguished a couple of services in our application and divided them on account of their responsibilities. The communication between them was designed in a way to keep the highest independence. We created the following microservices:

- meeting-service
- survey-service
- declaration-service
- chat-service
- notification-service
- web-application

To allow the web-application for the easiest communication with back-end services, we decided to use reverse-proxy in the form of Nginx, in which we defined the rules of requests distribution to every service. In order to ensure the indefectible information processing inside our system, we used the message queue - RabbitMQ [47]. Services communicate with each other with the help of REST API as well, through specially designed API available only in the network where all microservices operate. Each service has access to the database, and thus to the information it needs. The data of every one of them is isolated from other data, which enables further extension and modification by specifying the separate databases for every service.

The system architecture schema is presented below.

- The web-application sends the requests to reverse-proxy (Nginx [43]), which distributes the load between services.
- Services communicate with each other with the help of REST API and asynchronous messaging using RabbitMQ.
- Services use PostgreSQL [45] as the database.

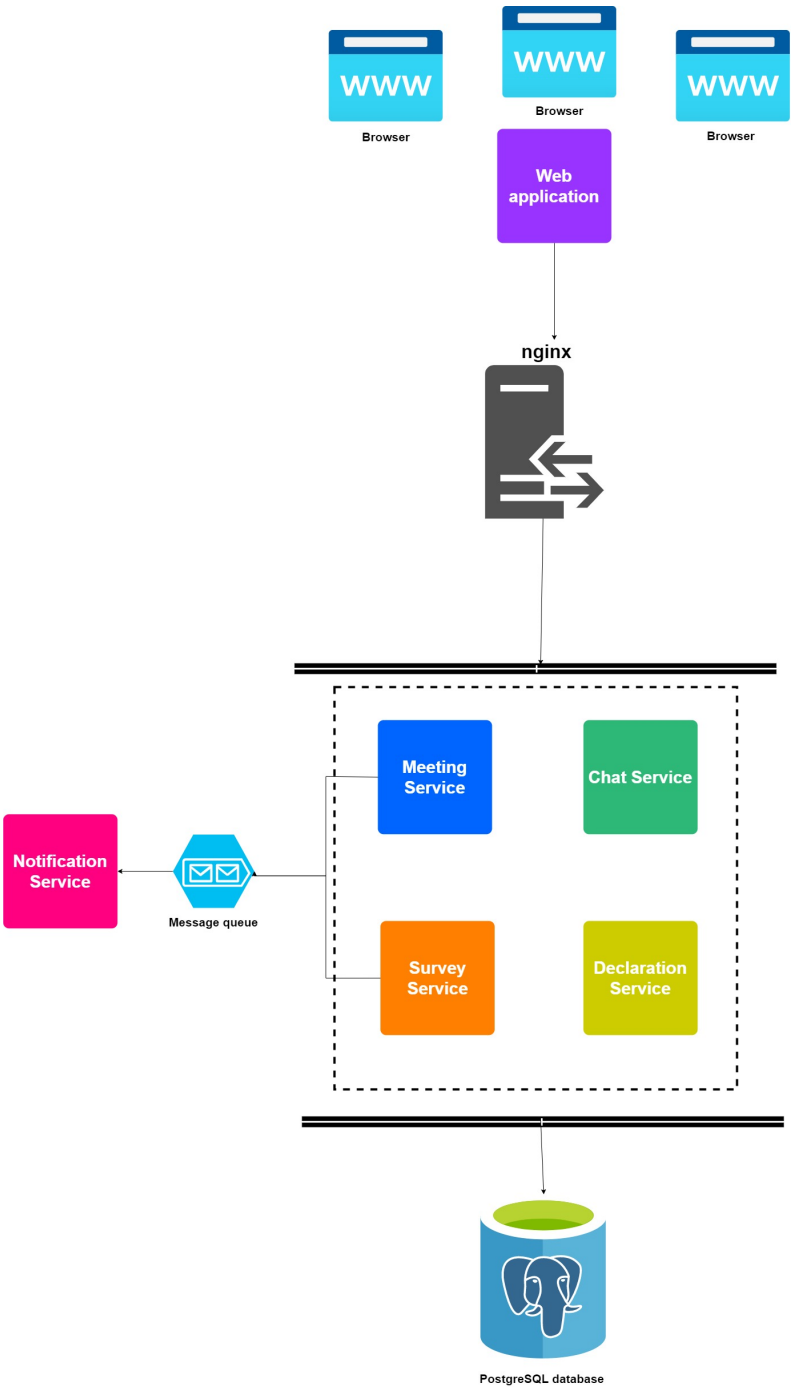

Figure 20: Communication schema between particular components

As mentioned before, we use REST API and asynchronous messaging to communicate within the group of services. The outline of the communication is presented below.

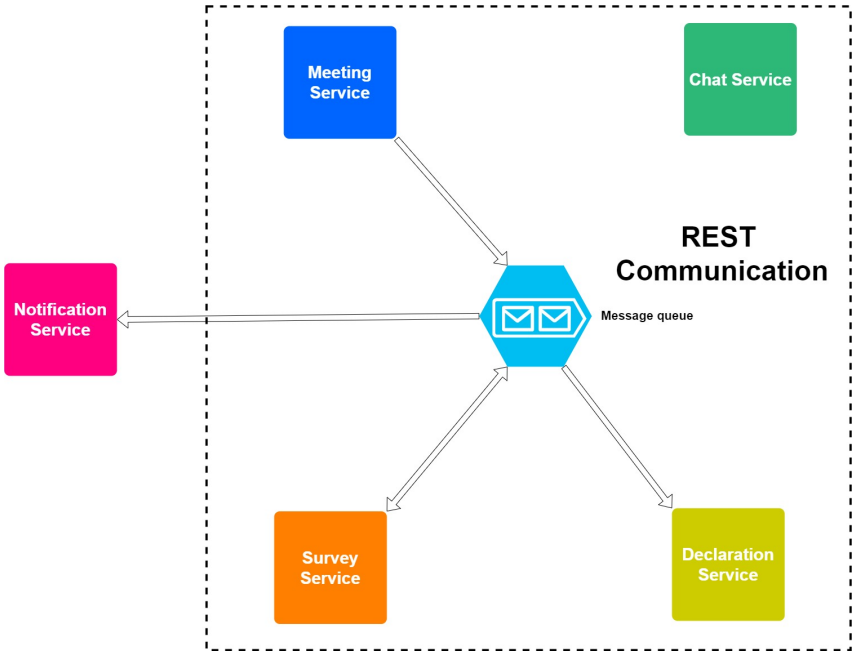

Figure 21: Communication schema between back-end microservices

REST API communication occurs in the group of four services because of the need of receiving responses from each other. Notification-service receives information only by the message queue and by REST API directly from web-application.

3.2. Technologies

3.2.1. Back-end

One of the first decisions we needed to make was the choice of a back-end services technology stack. Because of the architecture we adopted and resulting from this separation of microservices, we could let ourselves choose any language for each service independently. The technologies we decided to use are presented below.

3.2.1.1. Java

Java [25] is an object-oriented, statically-typed general purpose and the most popular for back-end applications programming language. It perfectly integrates with many existing libraries allowing rapid delivery of new features and solutions. We decided to use this language in the project, because of the biggest experience in Java. We knew from the beginning how to start the work with this.

There are following services written in Java in our system:

- meeting-service
- notification-service

3.2.1.2. Kotlin

Kotlin [31] is an object-oriented programming language developed by JetBrains. It eliminates the major cons of Java and is set on writing elegant, self-descriptive, and both object and functional code. This is the language that should crowd Java out of the market in the long term. At the beginning of the project, we reserved part of the time to learn new technologies. We selected Kotlin as we wanted to learn this language in a real project, get to know its advantages, and see how it solves some problems. The entry threshold was not high because we knew Java well before. However, we used Kotlin in smaller microservices so as not to risk.

There are following services written in Kotlin in our system:

- survey-service
- chat-service
- declaration-service

Different languages we considered are:

- Scala [51]
- Python [46]
- JavaScript [26]

We did not select Scala because of the high entry threshold and the lack of spring-boot integration. The next two languages were rejected due to the dynamic typing system, which as we think can be problematic and cause many problems in the long term.

3.2.1.3. Spring Boot

The next decision we needed to make is the framework choice, which should allow us to quickly develop back-end servers. The considered solutions are listed below.

- Spring-Boot [54]
- Micronaut [36]

We decided to use the first option because of prior experience with that framework and well-expanded documentation. The market share of Spring-Boot is much higher than Micronaut, and therefore there are plenty of materials on the internet describing the most common problems and solutions using this framework. Micronaut is quite a young solution nowadays, which does not contain rich education materials on the internet, so we had some concerns about using it.

3.2.2. Front-end

One of the most important decisions influencing the project and its final result was undoubtedly the choice of the main technology on the front-end side. The technologies we decided to use are presented below.

3.2.2.1. React

Due to the size and the team nature of the project, we rejected the implementations in pure JavaScript code, and instead, we decided to use something standardized, dedicated to our solutions. The choice had to be one of the three leading JavaScript frameworks: Vue.js [58], React [48], or Angular [4]. Based on our previous experience, we unanimously chose React. The main reason for making this decision was the fact that most of our team members have had prior experience with this technology. Considering that the visual aspect is a critical part of our project, we did not want to experiment and put effort into learning the new technology as it would be too risky. React fit the role of the main front-end tool perfectly, because it was proven, intuitive, and did not require any research-related workload. It is also worth mentioning that we wrote most of the React components ourselves, trying not to use external libraries. This is important as it introduces additional workload but allows us to create non-standard, dedicated components that increase the quality of the final product.

3.2.2.2. TypeScript and Prettier

The second important decision was to use TypeScript [57] and the Prettier code formatter. TypeScript is an overlay to JavaScript that introduces strong typing and Prettier enforces a uniform style of writing code. Both of these things introduce some additional effort related to formatting and explicit typing, but it benefits very quickly in the form of clean and consistent code. It turned out later to be a good decision that made it easier for us to evaluate and modify each other's code. These technologies turned out to be great in situations when we had to work with code that we wrote a long time before.

3.2.2.3. External libraries

The most important of the external libraries used in the project are Chart.js [7] and React-Bootstrap [49]. One of the objectives was to create a responsive application and we wanted to rely on something common and proven, so we used react-bootstrap to accomplish that task. The second very important library is chart.js, which allows us to create and customize the charts that we used to present the voting and survey results. The next library worth mentioning is material-UI [34]. We used only one component - DatePicker [9], that allows users to select the date and time, from this library.

3.2.2.4. Redux and state management

The key element of any web application is proper management of the application state. It has become popular to keep the state of the entire application in one place. This approach has many advantages such as providing easier debugging, ensuring a consistent flow of information, allowing us to go back to previous states, and eliminating a very important problem which is prop drilling. Prop drilling is a situation in which we pass a variable from a parent to a very nested child in the component tree. It increases a boilerplate code because all components on the path to the child have to pass a parameter to their child.

In the beginning, we chose Redux [50] to manage the state of the application, because it provides all the above-mentioned functionalities. After a number of the aforementioned advantages of this approach, it might seem that it is an ideal solution, but after a relatively short time (three or four sprints) we decided to remove Redux from our application. The decision was based on the fact that Redux introduced a lot of additional code (boilerplate code) and it has been designed for very large projects (like Facebook), so it turned out to be a too complex solution. Passing parameters to the nested components of our application is not that cumbersome, because in the worst case it requires passing parameters from parent to grandchild. The state of our application is therefore scattered across the components and is not stored in one place.

3.2.3. CI/CD

3.2.3.1. CI/CD process

The crucial part of the creation of web applications is the CI/CD process, by which we can ensure that changes we made in the source code do not destroy the previously working functionalities and the application deployment process in a production and development environment is easy and intuitive. We have no doubts that providing such a process in our application is one of the first challenges we needed to face. The CI/CD process existing in our project is shown below.

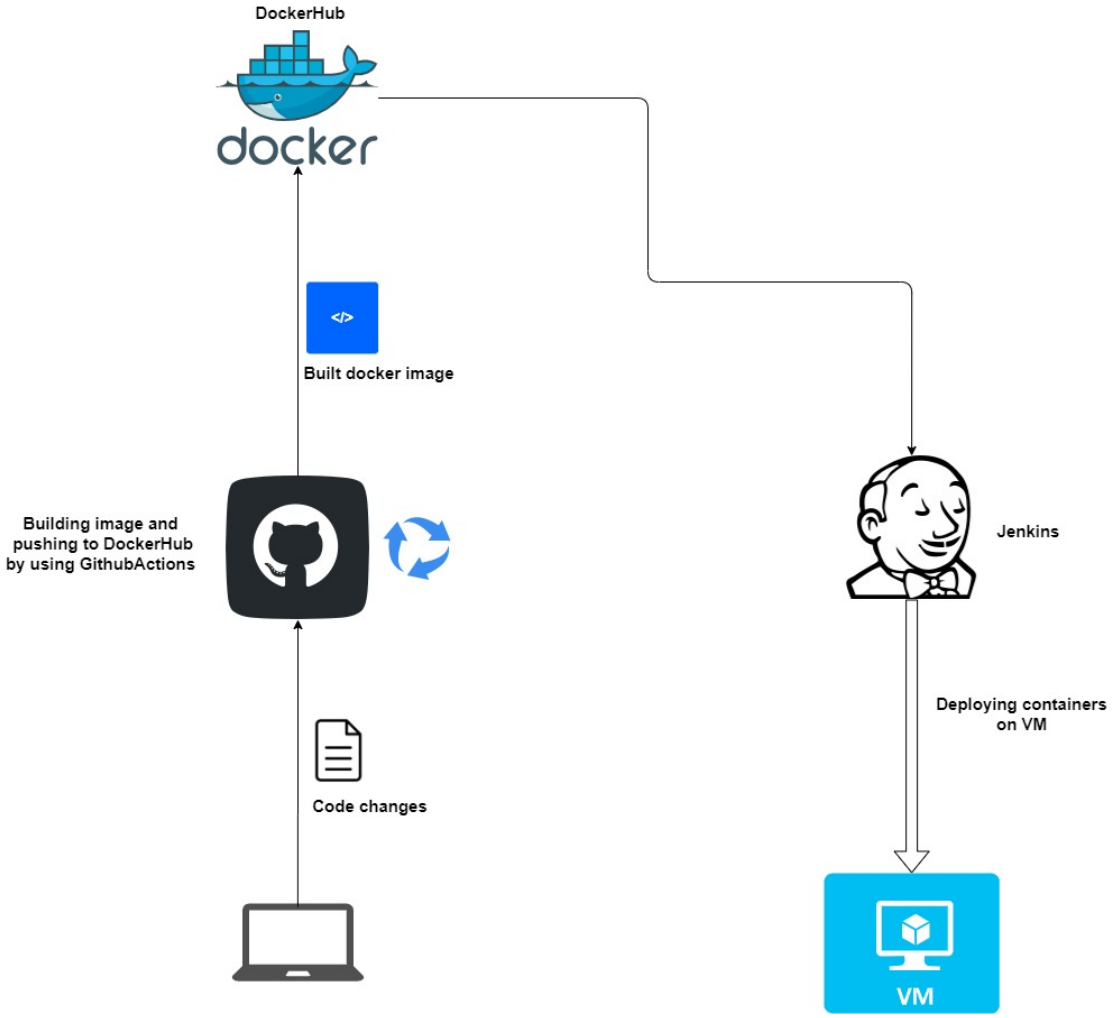

Figure 22: CI/CD process in our project

The above process contains few steps:

1. The developer pushes the source code changes from local to the global repository, which is GitHub [17].

2. When the changed source code appears in the repository, GitHub triggers an application build using GitHub Actions [18], during which the tests are run and a docker image is built. The image is built for one service as every service has its own repository.

3. When tests and build passes, the previously built image is pushed to DockerHub [12], which is the global repository of Docker [11] images.

4. Next with the use of Jenkins [27], we build the whole application, containing multiple microservices, with their dependencies (like Nginx [43], RabbitMQ [47]) on a Virtual-Machine.

We needed to make a couple of decisions while selecting the tools for the CI/CD process. The first decision was if we should use our own builds, defined as Jenkins Jobs, instead of the ones offered by Github Actions. We rejected such a possibility as the Gitbub Actions support is

so complex, that it completely covers all our needs with very little work from our side. Thanks to that we avoided writing all the “build” and “push to DockerHub” steps manually.

3.2.3.2. Jenkins

The next important aspect was the selection of the deployment tool. We needed to decide which of the following solutions will fit the best in our case.

- Jenkins [27]
- TravisCI [56]
- CircleCI [8]

We decided to use Jenkins as we knew this solution before. Additionally, Jenkins can be easily hosted on our own server in order to apply some custom actions to manage the environment. It was a very accurate choice as Jenkins has been used by us for managing the whole application and server, so it has made our processes automated.

3.2.3.3. Pipelines

Currently, there are three pipelines on our Jenkins, i.e. files that describe the steps that need to be executed. The pipelines commonly used by us to manage the application cycle in a production environment are presented below.

- **clean-server-from-images-and-old-containers** - this is a pipeline whose responsibility is to clear the production server from old docker images and the last of its, like volumes
- **pro-scheduler-application-setup-master** - pipeline used for deploying the newest version of the application on the server, which sets up the newest containers of each microservice and its dependencies
- **pro-scheduler-application-stop** - pipeline used to step the application running on the server

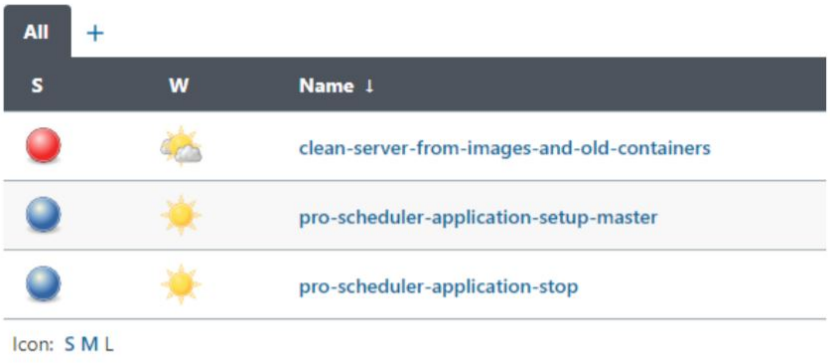

Figure 23: List of pipelines configured on Jenkins platform

3.2.4. Database

3.2.4.1. Relational database

The selection of the type of database is undoubtedly one of the most important steps in the application design process, therefore, before making the final decision, we conducted some research, collecting information about the available possibilities. Ultimately, in the project, we decided to use a relational database, which is characterized by a simple and intuitive way of presenting data in tables. The main factors in making this choice were the flexible way to store structured data, ensuring data consistency, and the ability to use the SQL query language.

The alternatives were graph databases such as Neo4j [42]. We decided not to choose such a solution, because our application is highly structured, does not have objects with strongly nested relations, and it would not fully use the potential of these databases.

Document databases such as NoSQL or MongoDB [39] have also been rejected due to low reliability. They provide access to collections that are not connected to each other, causing potential duplication of data. Moreover, verifying data integrity and consistency could be extremely difficult.

3.2.4.2. PostgreSQL

As a database management system, we have chosen PostgreSQL [45] version 13. The main advantage of this solution is its hybrid nature, i.e. the combination of a relational model with an object-oriented approach, which allows the use of abstractions such as inheritance. What is more, PostgreSQL is under an open-source license, so it is free and allows code modification. It is also the database most often chosen by developers due to its numerous functionalities, high extensibility and reliability.

MySQL [41], which is lightweight, fast and one of the most popular databases in the world, was also considered. However, its disadvantage was the lack of full compliance with the SQL query language and the problematic GPL license.

Another option under consideration was the use of MS SQL [40] - a database management system supported by Microsoft that enables data recovery. It was an interesting choice as we had previous experience with it during the third semester, however, it was not selected due to limited compatibility and license costs.

3.2.4.3. JPA

When writing about a database, it is important to mention about connecting to it and using it in the source code. For this purpose, we used a commonly known approach, which is JPA. It is an official object-relational mapping standard (ORM) that allows mapping the contents of a database to objects of the Java programming language. The most popular JPA implementation is the Hibernate [24] we use. Its advantages are good cooperation with Spring [53], clear code and a significant acceleration of the application development process. Although this approach may be slower than the JDBC standard which relies on a direct database connection using SQL, it reduces the considerable work involved in writing the code responsible for mapping individual database columns to Java classes.

3.3. Important mechanisms

3.3.1. Containerization with Docker

While developing an application based on microservices it is crucial to manage and deploy it in an easy way, which ensures the correctness of working. The microservice application can be complex because of the distribution of the system, and setting it up can be quite challenging. Due to the above, containerization of each service was a key, which we implemented at the very beginning. Without this, the setup of the application in both production and test environments could be enormously problematic. We used Docker, which is the most popular tool for containerization, whose biggest advantage is the possibility to push images to a global image repository - DockerHub. Such a solution will allow us to make a step further in the future and use Kubernetes to monitor and manage all our containers running in a cluster. We decided not to use Kubernetes so far, as this solution works best in a cloud environment and due to the limited budget, we used only a virtual machine to host and manage our application. Setting a Kubernetes agent on it would defeat the purpose and involve most of our server resources. The system we delivered is so elastic that in the future it is possible to move it simply to Amazon EKS [3] or another cloud provider, which offers Kubernetes [32] support.

3.3.2. Message queue

In the project, we used RabbitMQ [47] as the main element of asynchronous messaging between microservices. We considered other options like Kafka [30] or ActiveMQ [1], but in view of our biggest experience with RabbitMQ, we decided on this solution. In the future, it is important to consider changing it to Kafka, as this is a solution that scales the best and is destined for enormously big systems. Additionally, Kafka supports message replication and recovery after cluster member failure.

3.4. Application server

3.4.1. REST API in microservices

The achievement of microservice architecture was a huge challenge for us during the implementation. We needed to deeply consider what services should have been created, to avoid too fine-grained architecture, which could make further development harder due to many communication problems. Services have been divided in such a way that each of them covers a major functionality of a system. It minimizes the communication overhead and allows for rapid development.

Each of the microservices included in the system architecture provides the data via REST API. This exchange of information takes place mainly with the front-end application, however, some endpoints are dedicated to internal communication between individual microservices. All services with their responsibilities are presented below.

3.4.1.1. Meeting-service

Meeting-service provides REST API, which can be divided into below categories.

Meeting management (creating, deleting, updating meetings):

- **POST /api/meetings** - creating a new meeting
- **GET /api/meetings** - retrieving a list of all meetings the user is participating
- **GET /api/meetings/meetingId** - retrieving information about the meeting
- **PUT /api/meetings/meetingId** - updating information about the meeting
- **DELETE /api/meetings/meetingId** - deleting the meeting
- **POST /api/meetings/meetingId/leave** - leaving the meeting
- **POST /api/meetings/meetingId/cancel** - canceling the meeting

Meeting participants management:

- **POST /api/meetings/meetingId/attendees** - adding a participant to the meeting
- **GET /api/meetings/meetingId/attendees/attendeeId** - retrieving information about a given participant in the meeting
- **PUT /api/meetings/meetingId/attendees/attendeeId** - updating information about a given participant in the meeting
- **DELETE /api/meetings/meetingId/attendees/attendeeId** - deleting a given participant from the meeting

Meeting invitations management:

- **POST /api/invitations** - creating a new meeting invitation
- **GET /api/meetings/meetingId/invitations** - retrieving a list of all meeting invitations
- **GET /api/invitation**s - retrieving a list of all user invitations
- **POST /api/invitations/invitationId/accept** - accepting the invitation
- **POST /api/invitations/invitationId/reject** - rejecting the invitation
- **DELETE /api/invitations/invitationId** - removing the invitation

Users management:

- **GET /api/users/me** - retrieving information about the currently logged-in user
- **PUT /api/users/userId** - updating the information about a given user
- **DELETE /api/users/userId** - deleting the user

Authentication:

- **GET /api/auth/google** - callback for authentication via Google Sign-In
- **GET /api/auth/facebook** - callback for authentication via Facebook Login
- **GET /api/auth/logout** - logging out the currently logged in user

Google Calendar integration:

- **GET /api/auth/google-calendar** - callback for authentication and authorization with Google Calendar
- **GET /api/google-calendar/me** - retrieving information about the currently logged in user to Google Calendar
- **GET /api/google-calendar/me/calendars** - retrieving a list of calendars from Google Calendar from the currently logged in user

Geocoding support:

- **POST /api/geocoder/search** - displaying possible places of the meeting based on the searched phrase (facade for OpenStreetMap API)
- **POST /api/geocoder/cords** - retrieving information about the place based on coordinates data (facade for OpenStreetMap API)

Meeting places management:

- **PUT /api/places/meeting/meetingId/settings** - updating settings for voting at meeting place
- **GET /api/places/meeting/meetingId/settings** - retrieving information about place voting settings
- **POST /api/places/meeting/meetingId** - adding new places to the possible meeting places
- **PUT /api/places/meeting/meetingId** - updating possible meeting places
- **POST /api/places/meeting/meetingId/new** - adding a new place to the possible meeting places
- **GET /api/places/meeting/meetingId** - retrieving information about the possible meeting places
- **GET /api/places/placeId** - retrieving information about a given meeting place
- **PUT /api/places/placeId** - updating information about a given meeting place
- **DELETE /api/places/placeId** - removing a given meeting place
- **POST /api/places/placeId/vote** - voting for a given meeting place
- **DELETE /api/places/placeId/vote** - removing vote for a given meeting place
- **PUT /api/places/meeting/meetingId/votes** - updating votes for the possible meeting places

Shared link to the meeting support:

- **POST /api/share/generate** - generating a shared link to the meeting
- **GET /api/share/generatedEndpoint** - retrieving information about a given link to the meeting
- **POST /api/share/generatedEndpoint/join** - joining a meeting based on the shared link

Internal service endpoints:

- **GET /api/internal/meetings/meetingId/attendees-info** - retrieving information about the meeting participants
- **GET /api/internal/meetings/user/userId** - retrieving information about the user's meetings

- **POST /api/internal/meetings** - retrieving meeting information from the list of identifiers in batch mode
- **GET /api/internal/meetings/meetingId/settings** - retrieving information about the meeting settings
- **GET /api/internal/meetings/meetingId/user/userId/is-organizer** - retrieving information whether a given user is the organizer of the meeting

3.4.1.2. Survey-service

Survey-service provides REST API, which can be divided into below categories.

Survey management (creating, updating, filling surveys):

- **GET /api/surveys/user** - retrieving a list of all surveys in which the user participates
- **GET /api/surveys/surveyId** - retrieving information about the survey
- **POST /api/surveys** - creating a new survey
- **PUT /api/surveys/surveyId** - updating information about the survey
- **GET /api/surveys/meetingId/user** - retrieving information about the user survey for a given meeting
- **POST /api/surveys/fill** - filling the survey

Internal service endpoints:

- **GET /api/internal/surveys/surveyId/metadata** - downloading metadata about a given survey

3.4.1.3. Notification-service

Notification-service provides REST API, which can be divided into below categories.

Notifications management:

- **GET /api/notifications/meeting/meetingId** - retrieving the meeting notification settings
- **POST /api/notifications/meeting** - creating a meeting reminder
- **PUT /api/notifications/meeting** - updating a meeting reminder
- **DELETE /api/notifications/meeting** - removing a meeting reminder
- **GET /api/notifications/meeting/time/meetingId** - retrieving the settings for the reminder to vote for the meeting time

- **POST /api/notifications/meeting/time** - creating a reminder to vote for the meeting time
- **PUT /api/notifications/meeting/time** - updating a reminder to vote for the meeting time
- **DELETE /api/notifications/meeting/time/meetingId** - deleting a reminder to vote for the meeting time
- **POST /api/notifications/custom** - sending a custom message to meeting participants
- **POST /api/notifications/survey** - sending a reminder to complete the survey to all participants or to those who have not filled it yet
- **GET /api/notifications/survey/time/surveyId** - retrieving the settings for the reminder to fill the survey
- **POST /api/notifications/survey/time** - creating a reminder to fill the survey
- **PUT /api/notifications/survey/time** - updating a reminder to fill the survey
- **DELETE /api/notifications/survey/time/surveyId**- deleting a reminder to fill the survey
- **GET /api/notifications/settings** - retrieving participant preferences for receiving notifications
- **POST /api/notifications/settings** - updating participant preferences for retrieving notifications

3.4.1.4. Declaration-service

Declaration-service provides REST API, which can be divided into below categories.

Declarations management:

- **GET /api/declarations** - retrieving a list of all user declarations
- **GET /api/declarations/meeting/meetingId** - retrieving all declarations for a given meeting
- **GET /api/declarations/declarationId** - retrieving information about the declaration
- **POST /api/declarations** - creating a new declaration
- **PUT /api/declarations/declarationId** - updating information about the declaration
- **DELETE /api/declarations/declarationId** - removing the declaration
- **POST /api/declarations/declarationId/assign** - assigning to the declaration
- **POST /api/declarations/declarationId/unassign** - opting out from the declaration

3.4.1.5. Chat-service

Chat-service provides REST API, which can be divided into below categories.

Meeting chat support:

- **GET /api/chat/meeting/meetingId** - downloading chat messages for the meeting
- **POST /api/chat/meeting/meetingId** - adding new message to the meeting chat

3.4.2. Database schema

The diagrams of databases used by individual microservices are presented below. Due to the architectural characteristics, a given service refers only to its tables, knowing nothing about the tables of other microservices - this way the isolation of the database layer was emphasized.

The only exception is **the users table**, which stores information about users registered in the system, which is shared between services due to the designed architecture related to authentication.

3.4.2.1. Meeting-service

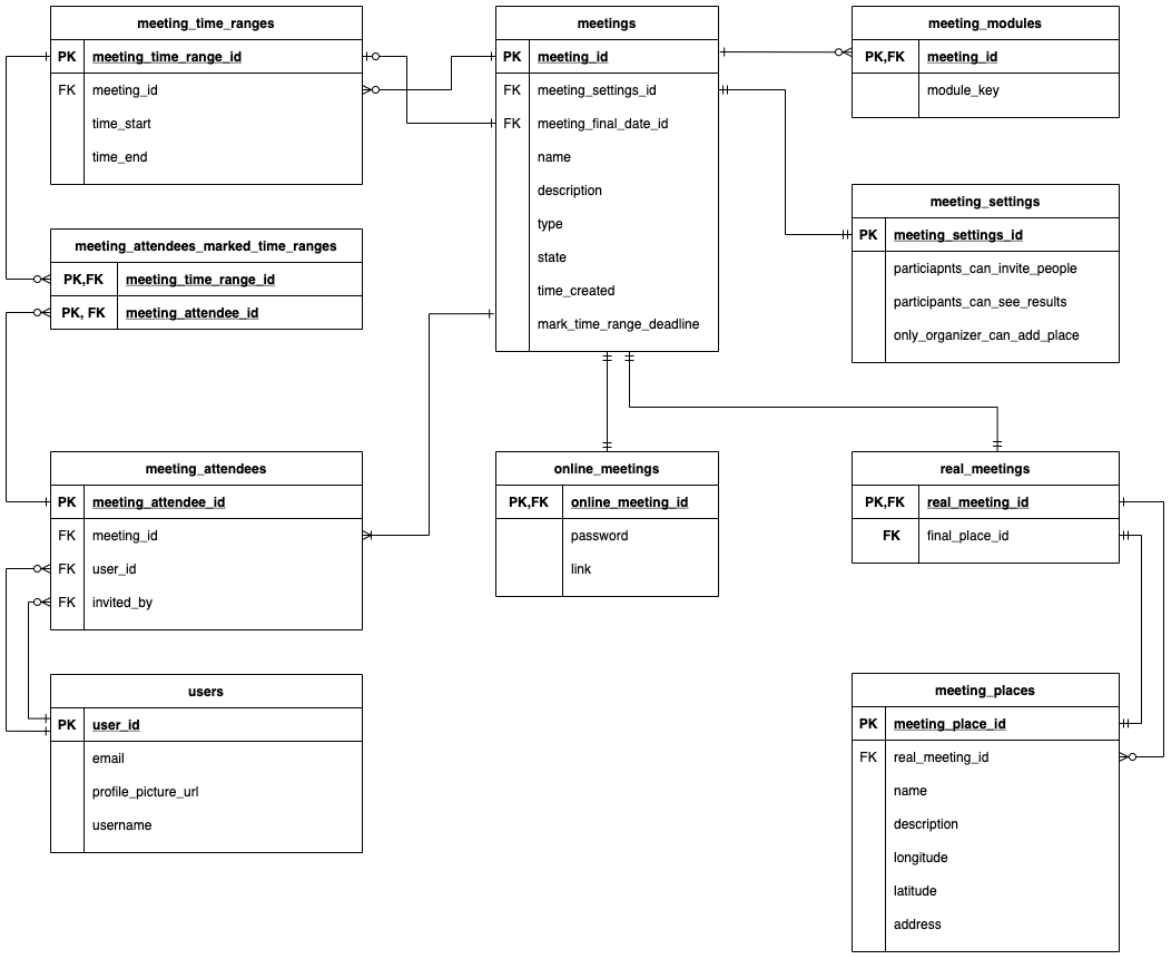

Figure 24: The database schema used by meeting-service

A detailed description of the tables used by meeting-service is presented below.

The meetings table stores basic information about meetings created by users.

- **meeting_id** - the meeting identifier (primary key)
- **meeting_settings_id** - the meeting settings identifier (foreign key)

- **meeting_final_date_id** - the meeting final date identifier (foreign key)
- **name** - the meeting name
- **description** - the meeting description
- **type** - the meeting type, specifying whether the meeting is real or online
- **state** - the meeting state, indicating whether the meeting has been canceled, is still open, or has ended
- **time_created** - the meeting creation time
- **mark_time_range_deadline** - the deadline by which the meeting participants can vote for the meeting date

The online_meetings table stores detailed information specific to online meetings. It is an extension of the information included in the meetings table.

- **online_meeting_id** - the meeting identifier (primary and foreign key)
- **password** - the online meeting password
- **link** - the online meeting link

The real_meetings table stores detailed information specific to real meetings. It is an extension of the information included in the meetings table.

- **real_meeting_id** - the meeting identifier (primary and foreign key)
- **final_place_id** - the final meeting place identifier (foreign key)

The meeting_modules table stores information about the modules attached to a given meeting.

- **meeting_id** - the meeting identifier (primary and foreign key)
- **module_key** - code name of the module

The meeting_settings table stores information about the settings of a given meeting, which determines the permissions of participants or the visibility of individual information.

- **meeting_setting_id** - the meeting settings identifier (primary key)
- **participants_can_invite_people** - a flag that determines whether meeting participants can invite new people to the meeting
- **participants_can_see_results** - a flag that determines whether meeting participants can see voting and survey results

- **only_organizer_can_add_place** - a flag that determines whether meeting participants can add new places as voting suggestions

The meeting_time_ranges table stores information about time intervals that can be voted or were selected by the participant.

- **meeting_time_range_id** - the meeting time range identifier (primary key)
- **meeting_id** - the meeting identifier to which the given time interval is associated
- **time_start** - the beginning of time interval
- **time_end** - the end of time interval

The meeting_attendees_marked_time_ranges table stores information about the voting time intervals selected by the meeting participant.

- **meeting_time_range_id** - the meeting time range identifier (primary and foreign key)
- **meeting_attendee_id** - the meeting attendee identifier (primary and foreign key)

The meeting_attendees table stores information about the attendees of the meeting.

- **meeting_attendee_id** - the meeting attendee identifier (primary key)
- **meeting_id** - the meeting identifier (foreign key)
- **user_id** - the user identifier (foreign key)
- **invited_by** - the identifier of a user who invited a given meeting attendee (foreign key)

The users table stores information about users registered in the system.

- **user_id** - the user identifier (primary key)
- **profile_picture_url** - the URL of the user's profile picture
- **username** - the user name

3.4.2.2. Survey-service

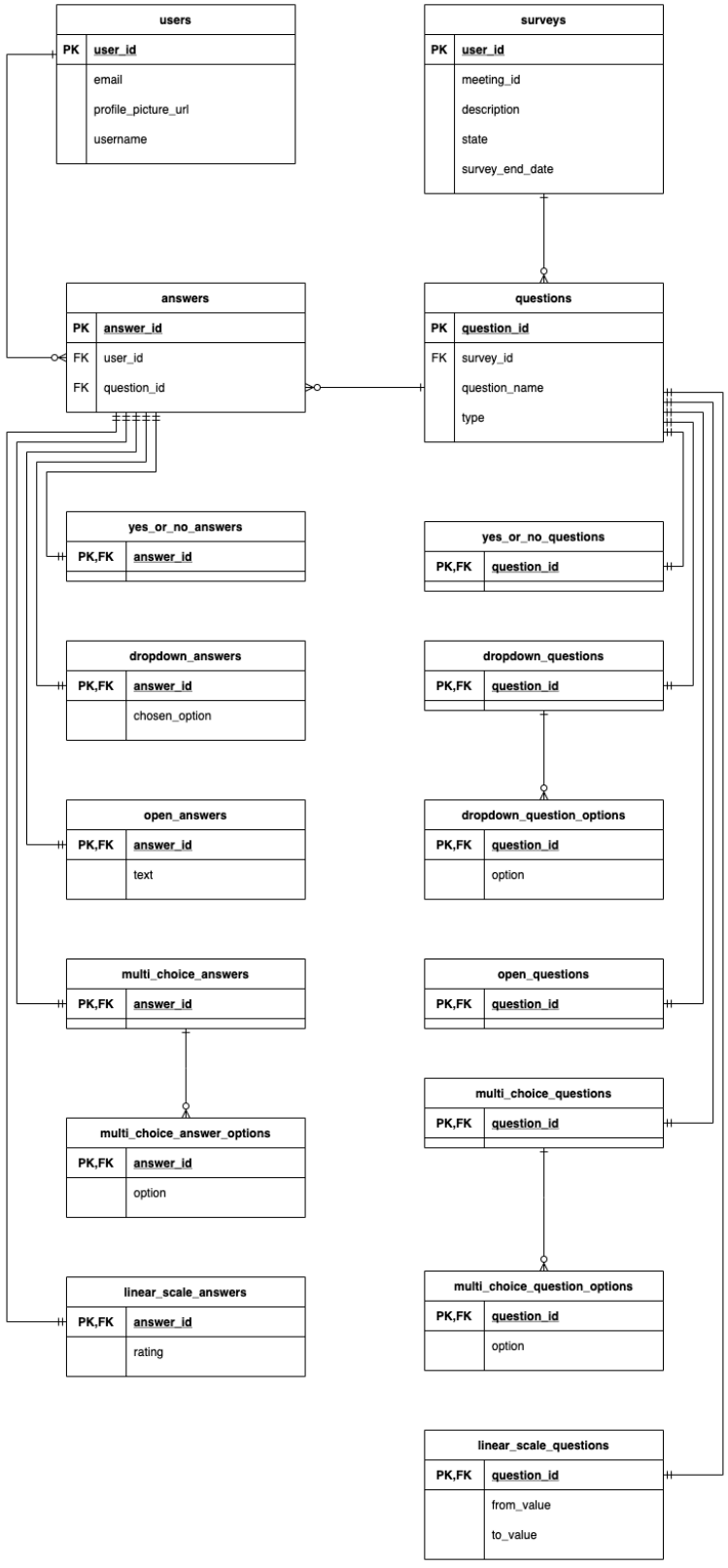

Figure 25: The database schema used by survey-service

A detailed description of the tables used by survey-service is presented below.

The surveys table stores information about the surveys created during the meeting.

- **survey_id** - the survey identifier (primary key)
- **meeting_id** - identifier of the meeting to which the survey relates (foreign key)
- **description** - the survey description
- **state** - the state of the survey, including whether meeting participants can complete the survey
- **survey_end_date** - the deadline by which meeting participants can complete the survey

The questions table stores basic information about the questions related to the survey.

- **question_id** - the question identifier (primary key)
- **survey_id** - the survey identifier to which the question relates (foreign key)
- **question_name** - the content of the question
- **type** - the question type

The answers table stores basic information about the answers to the survey questions.

- **answer_id** - the answer identifier (primary key)
- **user_id** - identifier of the user to whom the given answer relates (foreign key)
- **question_id** - identifier of the question to which the answer relates (foreign key)

The yes_or_no_questions table stores detailed information specific to YES/NO questions. It is an extension of the information contained in the questions table.

- **question_id** - the question identifier (primary and foreign key)

The dropdown_questions table stores detailed information specific to single-choice questions. It is an extension of the information contained in the questions table.

- **question_id** - the question identifier (primary and foreign key)

The dropdown_question_options table stores information about the possible choices for a given single-choice question.

- **question_id** - the question identifier (primary and foreign key)

- **option** - the single-choice question option

The open_questions table stores detailed information specific to open questions. It is an extension of the information contained in the questions table.

- **question_id** - the question identifier (primary and foreign key)

The multi_choice_questions table stores detailed information specific to multi-choice questions. It is an extension of the information contained in the questions table.

- **question_id** - the question identifier (primary and foreign key)

The multi_choice_question_options table stores information about possible choice options for multiple-choice questions.

- **question_id** - the question identifier (primary and foreign key)
- **option** - the multi-choice question option

The linear_scale_questions table stores detailed information specific to scale questions. It is an extension of the information contained in the questions table.

- **question_id** - the question identifier (primary and foreign key)
- **from_value** - the minimum value on the scale
- **to_value** - the maximum value on the scale

The yes_or_no_answers table stores detailed information specific to answers to YES/NO questions. It is an extension of the information contained in the answers table.

- **answer_id** - the answer identifier (primary and foreign key)
- **chosen_answer** - selected answer (i.e. YES or NO)

The dropdown_answers table stores detailed information specific to answers to single-choice questions. It is an extension of the information contained in the answers table.

- **answer_id** - the answer identifier (primary and foreign key)
- **chosen_option** - selected option

The open_answers table stores detailed information specific to answers to open questions. It is an extension of the information contained in the answers table.

- **answer_id** - the answer identifier (primary and foreign key)
- **text** - text of the answer

The multi_choice_answers table stores information specific to answers to multiple-choice questions. It is an extension of the information contained in the answers table.

- **answer_id** - the answer identifier (primary and foreign key)

The multi_choice_answer_options table stores information about the user-selected options contained in the multiple-choice questions.

- **answer_id** - the answer identifier (primary and foreign key)
- **option** - selected option

The linear_scale_answers table stores detailed information specific to answers to scale questions. It is an extension of the information contained in the answers table.

- **answer_id** - the answer identifier (primary and foreign key)
- **rating** - selected value from the scale

The users table stores information about users registered in the system.

- **user_id** - the user identifier (primary key)
- **profile_picture_url** - the URL of the user's profile picture
- **username** - the user name

3.4.2.3. Notification-service

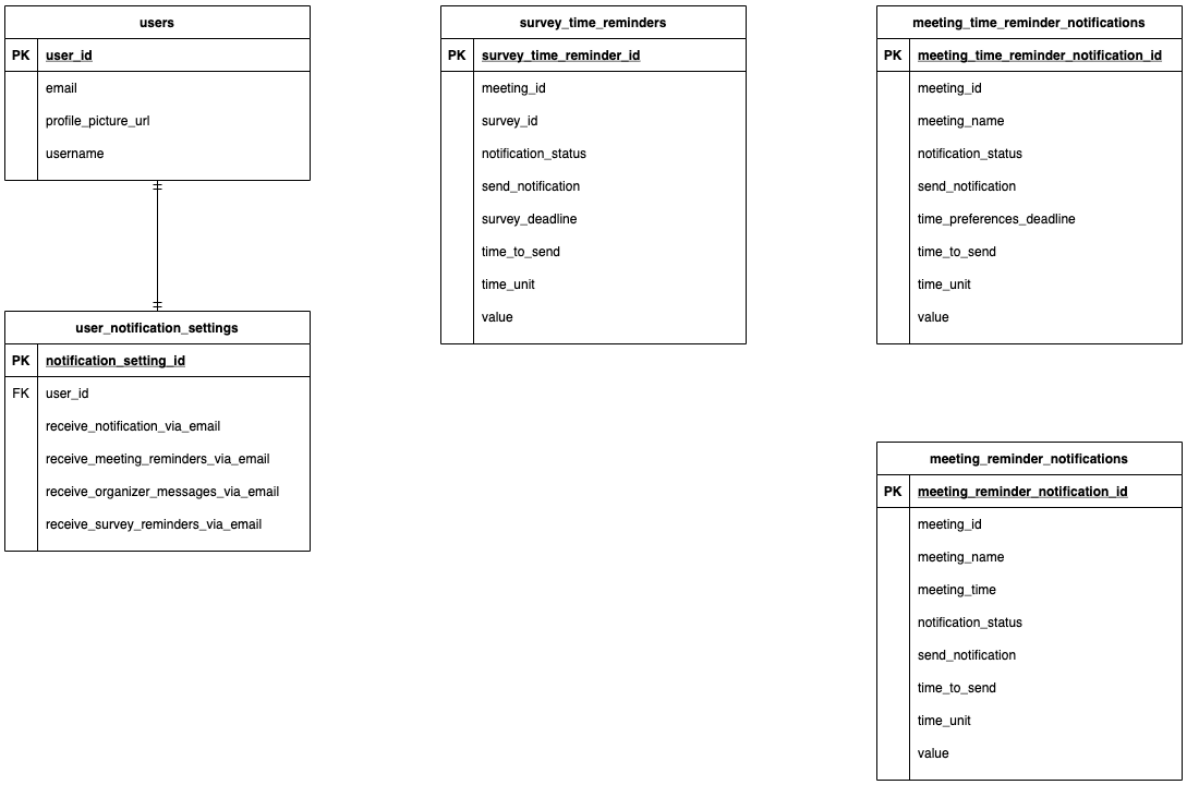

Figure 26: The database schema used by notification-service

A detailed description of the tables used by notification-service is presented below.

The user_notification_settings table stores information about user notification settings.

- **notification_setting_id** - the notification settings identifier (primary key)
- **user_id** - the user identifier (foreign key)
- **receive_notification_via_email** - a flag specifying whether the user wants to receive notifications via email
- **receive_meeting_reminders_via_email** - a flag specifying whether the user wants to receive notifications about upcoming meetings
- **receive_organizer_messages_via_email** - a flag specifying whether the user wants to receive notifications from the meeting organizers
- **receive_survey_reminders_via_email** - a flag specifying whether the user wants to receive notifications about the upcoming survey completion time

The survey_time_reminders table stores information about notifications related to the upcoming survey completion time.

- **survey_time_reminder_id** - the reminder identifier (primary key)
- **meeting_id** - identifier of the meeting to which the notification relates
- **survey_id** - identifier of the survey to which the notification relates
- **notification_status** - the notification sending status
- **send_notification** - a flag that indicates whether a given notification is active
- **survey_deadline** - the deadline for completing the survey to which the notification relates
- **time_to_send** - the date for sending the notification
- **time_unit** - time unit
- **value** - time unit value

The meeting_time_reminder_notifications table stores information about notifications related to the upcoming time preferences for filling.

- **meeting_time_reminder_notification_id** - the reminder identifier (primary key)
- **meeting_id** - identifier of the meeting to which the notification relates
- **meeting_name** - name of the meeting to which the notification relates
- **notification_status** - the notification sending status
- **send_notification** - a flag that indicates whether a given notification is active
- **time_preferences_deadline** - the deadline for completing time preferences for the meeting to which the notification relates
- **time_to_send** - the date for sending the notification
- **time_unit** - time unit
- **value** - time unit value

The meeting_reminder_notifications table stores information about notifications related to the upcoming meeting.

- **meeting_reminder_notification_id** - the reminder identifier (primary key)
- **meeting_id** - identifier of the meeting to which the notification relates

- **meeting_name** - name of the meeting to which the notification relates
- **meeting_time** - time of the meeting to which the notification relates
- **notification_status** - the notification sending status
- **send_notification** - a flag that indicates whether a given notification is active
- **time_to_send** - the date for sending the notification
- **time_unit** - time unit
- **value** - time unit value

The users table stores information about users registered in the system.

- **user_id** - the user identifier (primary key)
- **profile_picture_url** - the URL of the user's profile picture
- **username** - the user name

3.4.2.4. Declaration-service

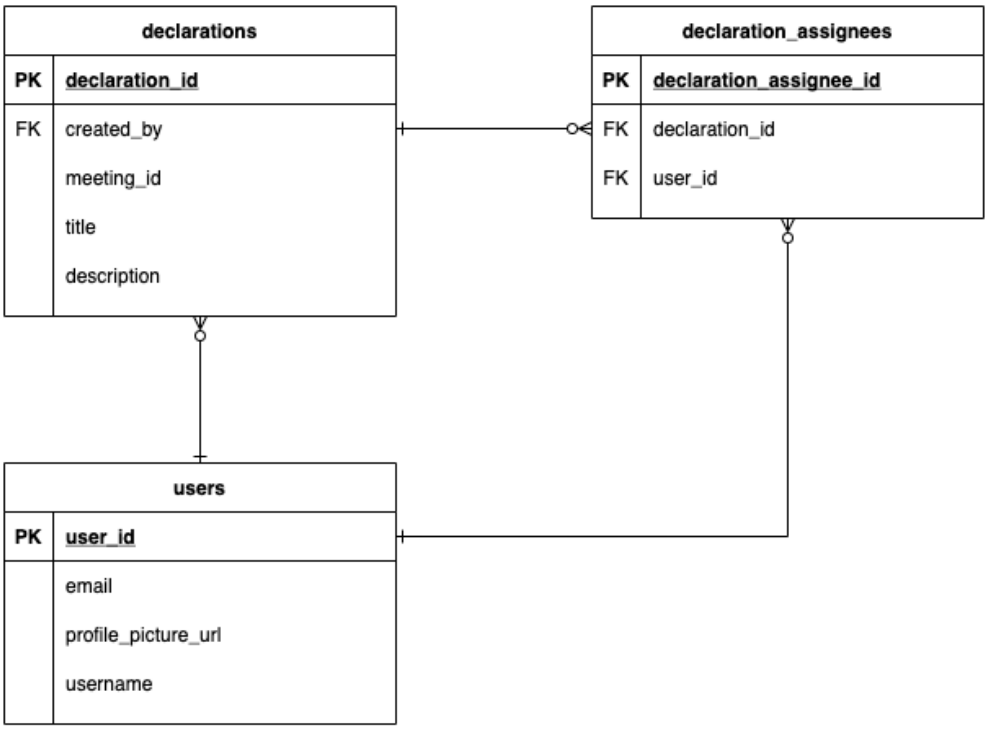

Figure 27: The database schema used by declaration-service

A detailed description of the tables used by declaration-service is presented below.

The declarations table stores information about declarations created for a given meeting.

- **declaration_id** - the declaration identifier (primary key)
- **created_by** - identifier of the user who created the declaration (foreign key)
- **meeting_id** - identifier of the meeting to which the given declaration relates
- **title** - the declaration title
- **description** - the declaration description

The declaration_assignees table stores information about meeting participants assigned to a given declaration.

- **declaration_assignee_id** - the meeting attendee identifier (primary key)
- **declaration_id** - the declaration identifier
- **user_id** - the user identifier assigned to the declaration

The users table stores information about users registered in the system.

- **user_id** - the user identifier (primary key)
- **profile_picture_url** - the URL of the user's profile picture
- **username** - the user name

3.4.2.5. Chat-service

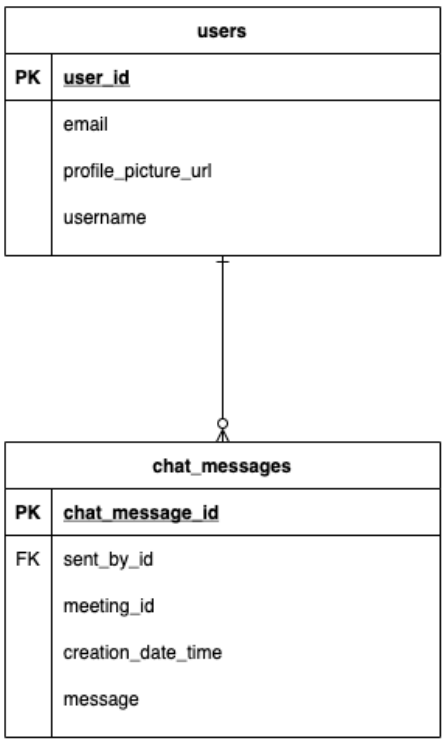

Figure 28: The database schema used by chat-service

A detailed description of the tables used by chat-service is presented below.

The chat_messages table stores information about messages sent in the chat of a given meeting.

- **chat_message_id** - the message identifier (primary key)
- **sent_by** - identifier of the user who sent the message (foreign key)
- **meeting_id** - identifier of the meeting to which the given message relates
- **creation_date_time** - time the message was created
- **message** - the message content

The users table stores information about users registered in the system.

- **user_id** - the user identifier (primary key)
- **profile_picture_url** - the URL of the user's profile picture
- **username** - the user name

3.4.3. Code structure

In order to organize the code in the best possible way, and thus work, the classes in each of the microservices have been grouped according to their functions and placed in individual packages. Among the most important, the following can be distinguished:

- **domain** - this package can be divided into three types of classes
  - classes corresponding to the relations in the database, using the annotations provided by Hibernate and
  - classes mapping REST queries and responses
  - classes representing repositories, enabling communication with the database services, i.e. classes responsible for the application operation logic
- **api** - contains classes marked with the @RestController annotation, i.e. controllers that enable receiving REST queries, forwarding them to individual services and returning responses
- **config** - a package containing classes with various configurations, such as connection to the database, user authentication when logging in with Google or Facebook, parameters of queues used for notifications, or all types of security, including CORS, CSRF, authorization at the login level and possible parameters of REST queries.

3.4.4. Tools and dependencies

Each of the microservices uses all kinds of tools and dependencies to facilitate the software development process.

3.4.4.1. Maven

Maven [35] is a tool that automates the software development for the Java platform, which allows you to find and download all kinds of dependencies and libraries in an orderly and intuitive way. It also makes it easy to compile, test, and build the project. In our application, Maven is used by two microservices written in Java, i.e. meeting-service and notification-service. Each of them contains a pom.xml file, where all necessary dependencies are defined.

3.4.4.2. Gradle

Gradle [23] is also a project-build tool that downloads the dependencies specified in the build.gradle file. It was used by us in microservices written in the Kotlin language. So it is used by chat-service, declaration-service and survey-service.

3.4.4.3. Lombok

Lombok [33] is a Java library that provides all kinds of annotations, allowing for automatic code generation, and thus facilitating and accelerating class definition. It allows, among other

things, to generate getters, setters and constructors, thanks to which we do not have to implement them ourselves, saving time. Moreover, the code becomes more transparent and readable.

3.5. Front-end application

3.5.1. Code structure

The code structure of the front-end application is in a way enforced by the React framework, which is based on components. The created directories are shown below.

- **views** - all application views, such as landing page, meeting details view, etc.
- **components** - custom components used through the application in various views, such as buttons, calendars, etc.
- **CSS/tokens** - custom styles, colors, and CSS animations
- **tools** - own Hooks, i.e. special React functions and form validation functions
- **model** - structured TypeScript types mapping the domain model and the model of communication with the back-end. Due to enforcing explicit typing by TypeScript, in this folder, there are stored all kinds of server queries and responses
- **auth** - functions needed for user authentication
- **API** - functions and URLs responsible for communication with individual microservices

3.5.2. Communication with the application server

As mentioned, the communication model between the client and the server is based on the REST API interface. Sending HTTP requests is therefore a key system functionality that can be provided by external libraries such as Axios. However, we chose not to use any of them, but to use the built-in fetch function. We have tried to simplify sending requests as much as possible by creating generic functions covering all basic HTTP methods (PUT, GET, POST, DELETE), so that the process is limited to the selection of the method, URL, and possibly transmitted data. All the rest, i.e. adding authentication headers, handling responses and errors, is provided by generic code. Moreover, thanks to the use of reverse-proxy, the front-end application does not need to know the exact location of specific microservices, because the distribution of queries is entirely handled by Nginx, which enables complete isolation of the front-end from the back-end.

3.6. Graphics

The appearance of the final product is very important in achieving customer satisfaction. Therefore, we started designing the UI of our application with the selection of the graphic layout.

At the beginning we focused on a color palette. We tried to choose the shades that are eye-friendly and interesting at the same time. As a result, the main color present in the details of individual elements are shades of purple, standing out against a whitish background. On the other hand, all types of buttons have been designed in green tones that will attract the attention

of potential users and give a subconscious signal to click the selected element. The color palette, due to its delicate shade, is also intended to create a pleasant and calming mood.

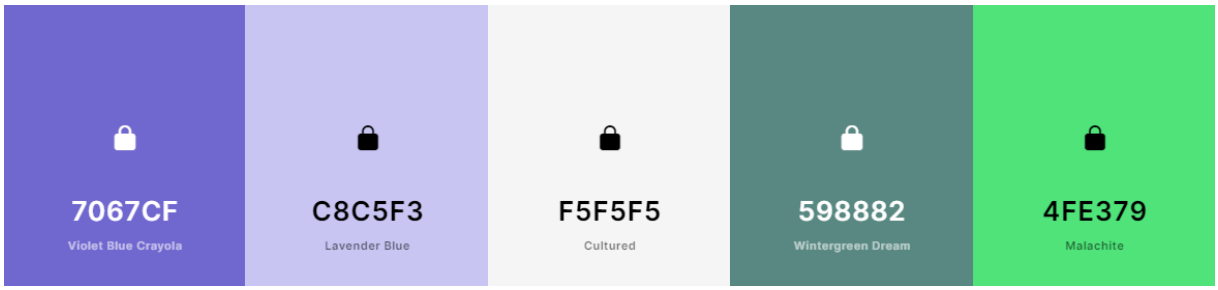

Figure 29: The color palette used in design of the front-end application

The next stage of the project was the creation of mock-ups. For this purpose, we used the AdobeXD program [2], with which we had some previous experience. This tool is used to create interactive prototypes of mobile and web applications. Its main advantage is the possibility of working on one project by many people at the same time, which significantly speeds up and facilitates teamwork. We have finally designed three versions of the application throughout the entire project, introducing changes with future users of the system in mind and based on the client's suggestions. Diligence, intuitiveness, transparency and user-friendliness were the key criteria that we constantly paid attention to.

3.7. Quality Assurance

A very important part of any software development is simultaneous testing of changes in order to prevent possible damage to functionalities in the future. In fact, this is the only way to keep code safe from introducing a bug by the developer that disturbs the application. Due to the dynamic development of our project and adding newer and newer functionalities, this aspect was especially important for us. However, due to our focus on creating as many features as possible during the project's development, only the most essential elements have been tested so far. Two types of tests can be distinguished in our application:

- unit tests
- integration tests

The former was used to ensure the proper operation of individual system components, while the latter was used to test integration within the service. We used the following libraries to test our services:

- **JUnit5** [29] - a Java library that provides assertions mechanisms
- **Mockito** [38] - library used for mocking object dependencies
- **spring-boot-starter-test** - provides the API for verifying the Spring-Boot applications

The development of the test infrastructure ensured application correctness verification during the whole development process and allowed us to catch a few accidentally introduced errors.

3.8. Faced challenges

3.8.1. Back-end

During the development of the application based on microservices, we faced a couple of problems on the back-end side of the system. This kind of architecture is much more challenging than the monolithic one, as it requires a different approach for some topics like communication in the system and consistency ensurance. The data access is restricted due to its distribution, which implies the necessity of communication between services.

We implemented two ways of communication:

- REST API
- asynchronous messaging using queue

The first kind of communication is used when the service needs a response from the service that asks, e.g. while displaying the list of unfilled surveys to load the meeting names. In case when the answer is not needed and the guarantee of message delivery is crucial, we use the second approach - queue. It guarantees reliability with a proper configuration and is able to recover from the unavailability of the receiver site.

The next problem we faced is related to time zones. As our application is designed for global use, it was mandatory for us to handle all time zones and avoid inconsistency. On the back-end side, all the dates are stored in Java Instant format that represents a moment of time in a standardized time zone. There is no information on the back-end side from which localization the request comes from, therefore the front-end is responsible for converting the standardized date into a specific time-zoned date for given localization.

The next technical problem was to properly handle the resources on the server we bought. We had a limited number of resources and needed to run many applications like microservices and Jenkins. It was a challenge to fit all of these components into one virtual machine, but due to Docker features like container memory usage limitations, we could successfully manage the server.

3.8.2. Front-end

3.8.2.1. Component for selecting time intervals

One of the biggest challenges on the front-end side was creating a component for marking time ranges. The problem consisted of numerous sub-problems because it was necessary to take into account such issues as:

- possibility of extending and narrowing the compartments vertically
- protection against going beyond the given time range
- possibility of shifting selected intervals
- effect of combining them in the event of overlapping
- correct display of hours on the component's label and on the hour grid.

We tried to find a ready-made solution, but unfortunately, nowhere did we come across a component that would meet the given criteria.

Ultimately, we decided to use the appropriate positioning attributes of HTML elements and two external components, such as Resizable from the re-resizable library, and Draggable from the react-draggable library. These components made it possible to manipulate the size of the compartments. To generate a grid of hours, we created twenty-four buttons, setting both their position attributes to the “relative” value and a constant height. It was also important to choose the right z-index attribute so that the buttons on the hour grid did not cover the time intervals. To combine many time slots, we used the useEffect function from the React library, which listens for changes in intervals and checks if any of them intersect each other. The obtained solution is presented below.

Figure 30: Components for selecting time intervals

The green compartments can be moved to any place, their length can be changed and they can be joined together. The cross in the upper right corner of each is used to delete them.

3.8.2.2. Generic nature of HTTP requests and error handling

Another difficult technical problem was the adequate design of HTTP requests to the back-end application so that errors and the query processing status could be properly displayed. The

query can be in one of the three states:

- **loading** - the query has been triggered and we are waiting for a response from the server. Storing this state allows us to display a spinner component so that the user can see that the action taken is being processed.
- **success** - the query has been positively processed. In this state, we receive proper data from the server and we can also display a message about the successful completion of the action.
- **failed** - the query was not processed successfully. In this state, we can react to the received error and inform the user in an appropriate manner about its occurrence.

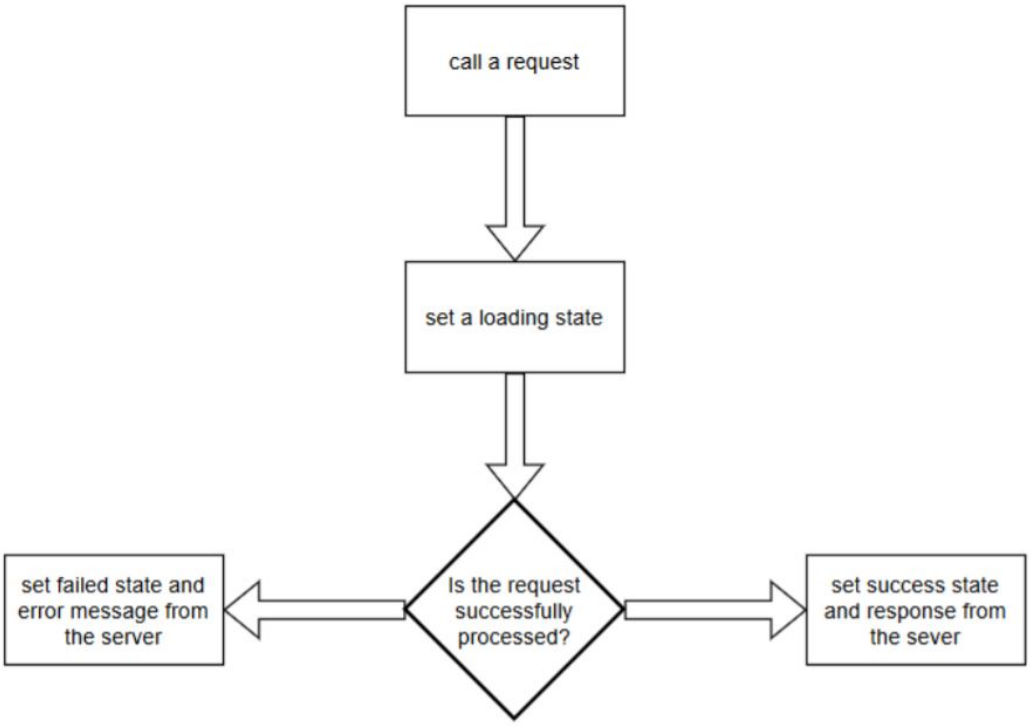

Figure 31: Processing HTTP requests on the front-end side

The whole difficulty of the above problem is the variety of queries. There are factors such as choosing the appropriate HTTP method and whether or not to send data in the request. What is more, each request may have a different URL, not everyone will return data (the so-called fire-and-forget strategy), for some of them, we may want to display a message, and not for others. All of the above factors contribute to the difficulty of properly designing requests to the REST API server, so as to minimize the "boilerplate code". The solution we found is as follows:

- a separate file with corresponding endpoints is created for each microservice, which allows them to be managed in an optimal and legible way
- for each possible action, a new function is created, thanks to which calling a query in any React component comes to calling its name

- for each type of request, generic functions are created, which additionally manage its state, using the mechanism of promises built into JavaScript

3.8.2.3. Map scaling

The last technical problem worth describing is the correct positioning and zooming on the map. As we know, our application allows users to vote for several places on the map, so we wanted all possible suggestions to be visible at a glance. In the first step, we counted the point being the center between the extremely distant points on the map, separately for latitude and longitude. In the second step, using the trial and error method, we set the relevant zoom of the map so that its surface covers all the extreme points.

4. Work organization

4.1. Characteristics of the project and the way of its implementation

From the very beginning (i.e. from the first meetings with the client), the project had precisely defined high-level goals, which allowed to establish a work plan both in the first semester, during the summer break, and in the second semester. In addition, being aware that some aspects of the application during their implementation may turn out to be more complex and difficult, individual stages of realization have been planned to take into account the time reserved for possible difficulties.

4.1.1. Work progress over time

The work schedule assumed at the beginning of the project was maintained with an accuracy of one month. The main planned and implemented stages of work through the duration of the project are presented below.

Summer semester 2020/2021

- **March 2021** - getting acquainted with the subject of the project and defining the overall vision (Phase I)
- **March/April 2021** - selection of technologies and preparation of tools (Phase II)
- **April/June 2021** - implementation of the prototype (Phase III)

Summer break

- **July - September 2021** - verification of the prototype and implementation of other functionalities (Phase IV, part I)

Winter semester 2021/2022

- **October - November 2021** - verification of the prototype and implementation of other functionalities (Phase IV, part II)
- **November 2021** - product testing and bug fixes (Phase V, part I)
- **November/December 2021** - product and documentation finalization (Phase V, part II)

4.1.2. Software development methodology

The methodology that we decided to use to implement our project is a modified version of Scrum. The main assumptions were retained, such as the division into two-week sprints, which were always preceded by the planning of new iteration and assigning points to individual tasks. However, we resigned from daily meetings in favor of meeting with all the team members once every two weeks. There were also independent meetings of individuals, during which mainly issues related to the implementation of specific tasks were discussed. The methodology created in this way was dictated mainly by the fact that regular daily meetings turned out to be impossible due to other duties of individual team members, such as varied study schedules or other additional activities.

Therefore, in fear of a possible loss of synchronization between team members due to the lack of the aforementioned daily meetings, it was extremely important to choose the right tool for managing the software development process in the Scrum methodology. We focused on proven solutions such as the Jira platform, which helped us maintain the continuity of work and an appropriate information flow in the team.

As a result of the adopted assumptions regarding the organization of work - the whole development process of the entire product is based on the iterative model with increments. This type of model fits perfectly to all of the project requirements as the general vision of the product with some more specific details has been presented from the early beginning stage of the development but there was still possibility to introduce some changes to how the final product should look like.

4.1.2.1. Planning meetings

As mentioned before, the weekly planning meetings had a very important role in ensuring proper synchronization of the entire team - mainly due to the lack of daily meetings assumed at the beginning of the project implementation, which is one of the main elements of the Scrum methodology.

Each planning meeting was held via the Microsoft Teams platform in the form of an online meeting with all team members. During the meeting, we were discussing the results achieved during the previously completed sprint. Based on the results and the set of goals we selected new functionalities to implement, which were then divided into smaller tasks. Next, the estimation and distribution of points for individual tasks took place, and finally, the ability to perform them was verified during the iteration (i.e. two weeks). During each meeting, the person responsible for planning provided a screen with a board view on the Jira platform so as to effectively carry out the next stages of realizing the project.

4.1.3. Division of responsibilities

From the very beginning of the project, the division of work on individual aspects of the product was created in a natural way, due to the preferred technological facilities of each of the team members. Nevertheless, in the case of the greater overhead of work on one component in comparison with the rest of them, others also participated in their implementation. This allowed for

dynamic adaptation of the software development, which resulted in a steady increase in new functionalities throughout the entire period of the project.

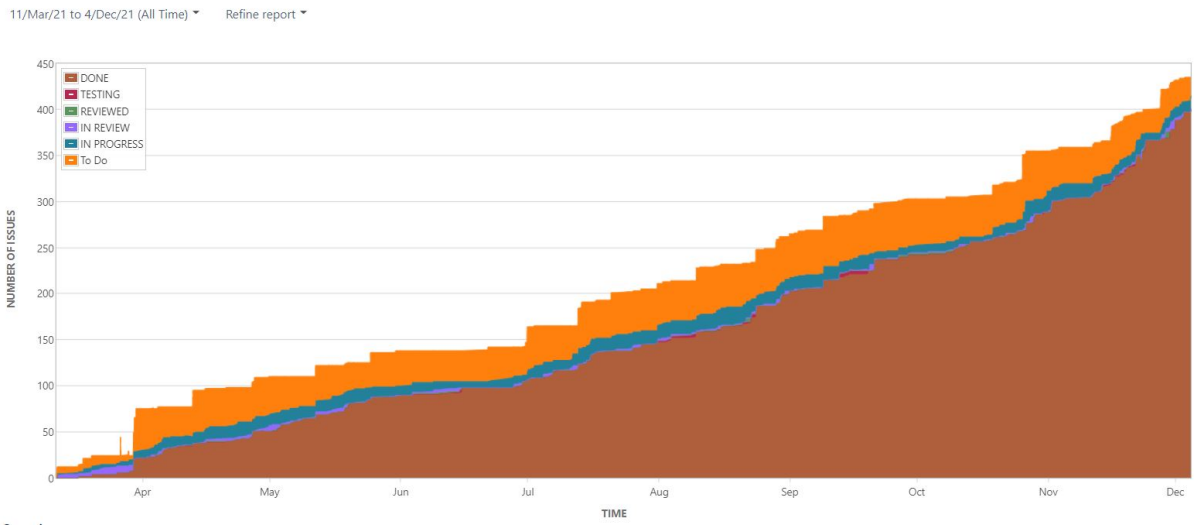

Figure 32: A graph showing the steady growth over the entire project implementation period (here: the number of tasks performed on the Jira platform)

4.1.3.1. Implementation of tasks - Kopeć Radosław

The list of tasks for which Radosław Kopeć was mainly responsible is presented below.

- implementation, development and maintenance of the front-end side of the application
- creating components, views and implementing communication with the server within the time module
- creating components, views and implementation communication with the server within the declaration module
- creating components, views and implementation communication with the server within the place module
- designing generic communication with the back-end layer
- creating REST API documentation for declaration and place modules
- creating UI/UX mockups reflecting the vision of the project using AdobeXD
- visual improvements to both UI 2.0 and the final version, taking into account the opinions from UX research
- creating reusable components
- code refactor to remove Redux
- constantly ensuring responsiveness of the application

- fixing bugs on the front-end side
- adding a custom color theme

4.1.3.2. Implementation of tasks - Mazur Aleksandra

The list of tasks for which Aleksandra Mazur was mainly responsible is presented below.

- implementation, development and maintenance of the front-end side of the application
- creating basic and reusable components in the user interface, such as buttons, icons, and forms
- creating the landing page and user profile view
- implementation of the meeting creation flow on the front-end side
- creating components, views and implementing communication with the server within the survey module
- creating components, views and implementing communication with the server within the notification module
- creating views with list of meetings, surveys, declarations and invitations
- front-end management of meeting participants (sending invitations, removing participants, leaving the meeting, assigning the role of the organizer, etc.)
- implementation of the functionality of sharing a meeting via link on the front-end side
- visual improvements to both UI 2.0 and the final version, taking into account the opinions from UX research
- code refactor to remove Redux
- constantly ensuring responsiveness of the application
- fixing bugs on the front-end and back-end side
- management of documentation creation

4.1.3.3. Implementation of tasks - Surjak Mateusz

The list of tasks for which Mateusz Surjak was mainly responsible is presented below.

- implementation, development and maintenance of the back-end side of the application
- creating an email notification system
- implementation of the back-end part of the chat
- implementation and development of survey-service

- development of the functionalities of meeting-service and declaration-service
- implementation of geocoding functionality
- implementation of the functionality of sharing a meeting via link on the back-end side
- containerization of the application
- integration and implementation of CI/CD in the system
- creating Jenkins pipelines
- project management from the DevOps side
- maintenance of the production server
- integration of individual components of the back-end side
- source code repositories setup and integration with GitHub Actions
- development and maintenance of the system architecture
- designing the REST API for individual microservices (survey-service, chat-service, notification-service)
- fixing bugs on the back-end side
- implementing Nginx reverse-proxy

4.1.3.4. Implementation of tasks - Zielonka Marcin

The list of tasks for which Marcin Zielonka was mainly responsible is presented below.

- creating a login process via Google and Facebook
- creating and maintaining internal libraries used by other components of the project
- configuration of a private Maven repository using Artifactory and connecting it with individual project components (including configuration of the repository in the CI process)
- creating a meeting chat
- creating REST API documentation for individual microservices for internal needs in the project
- creating UI/UX mockups reflecting the vision of the project using AdobeXD
- organizational project management on the Jira platform
- conducting internal planning meetings at the beginning of each sprint
- designing the REST API for individual microservices (meeting-service, declaration-service)
- refactoring of the meeting-service in order to standardize the REST API

4.2. Project management tools

The scope and character of the project forced us to use many professional frameworks and tools. We have chosen them very carefully to be able to efficiently and seamlessly manage our project during various stages of implementation. These tools can be divided according to their purpose and functionalities as follows:

- planning
- source code management
- team communication

4.2.1. Planning

4.2.1.1. Jira

The main tool used to organize short-term (individual sprints) and long-term (milestones) work was the Jira [28] platform created by Atlassian. We chose a cloud-based version of Jira software instead of the On-Premise software hosted by the University because of its flexibility and ease of configuration.

The project on the platform has been divided into two parts: the part related to the planning of individual sprints and the part responsible for tracking the progress at the level of individual user stories. User stories have been developed in cooperation with the client at the beginning of the project implementation.

Part one - Scrum:

We have accomplished seventeen two-week sprints during the entire project implementation period (i.e. from March 11, 2021, to December 5, 2021). Each sprint was preceded by an internal team meeting during which the planning took place. Every planning included:

- defining the goal of a given sprint
- selection of user stories to be implemented in the sprint
- division of user stories data into smaller tasks
- distribution of points (User Points) for each task
- the initial division of work between team members

Each of the sprints contained tasks whose total sum was approximately 50 Story Points on average. When analyzing further sprints, it turned out that this is the optimal overhead of tasks for a given cycle, which meant that the progress of work throughout the entire project implementation period was relatively equal and - most importantly - continuous.

In order to effectively achieve the following goals, the sprint table has been divided into the following columns:

- To Do
- In Progress
- In Review
- Reviewed
- Testing
- Done

Such a division allowed for easy communication between individual team members, as well as for mutual verification of the implementation of single tasks by reviewing the solutions. Worth mentioning are columns “In Review” and “Testing” because they always helped us to more easily recognize that our solutions do not require further work.

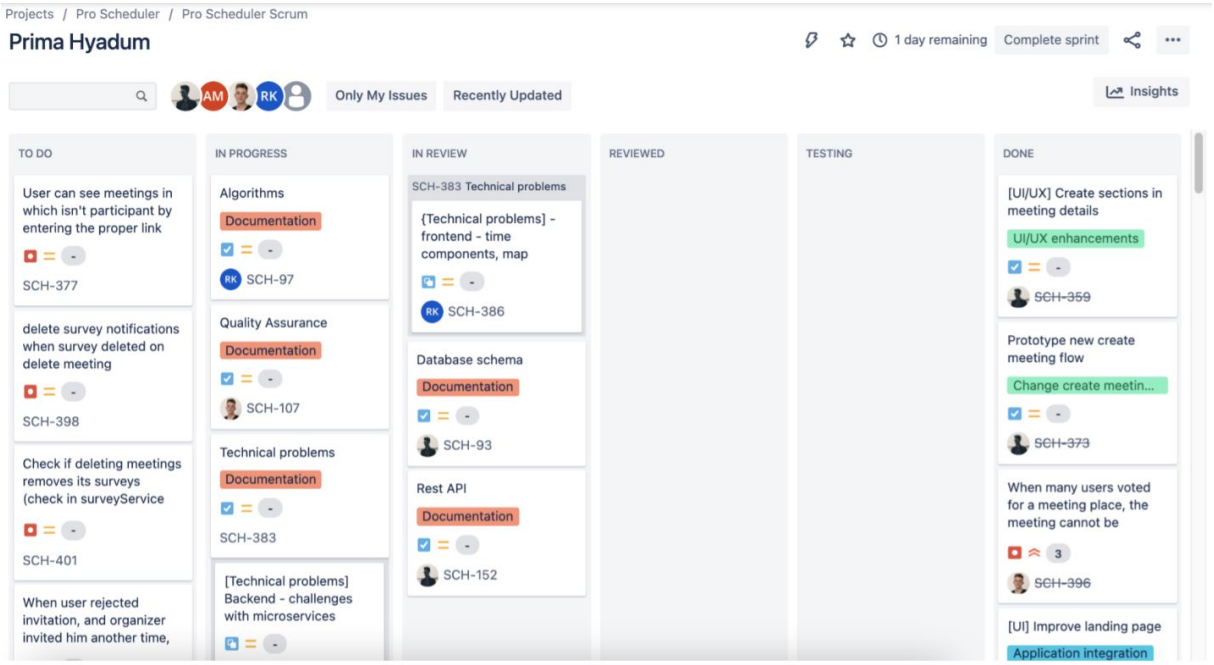

Figure 33: The board used to organize a sprint on the Jira platform

Part two - Epics:

An important part of planning and managing the project from a long-term perspective was the creation and maintenance of a second dashboard that reflected the priorities of the individual user stories. Epics have been divided into columns:

- Would
- Could
- Should

- Must
- Next Sprint
- Current Sprint
- Done

Such a division allowed for the effective selection of appropriate tasks as part of subsequent sprints and for tracking their progress. It was very important that the client knew about the total progress of a project.

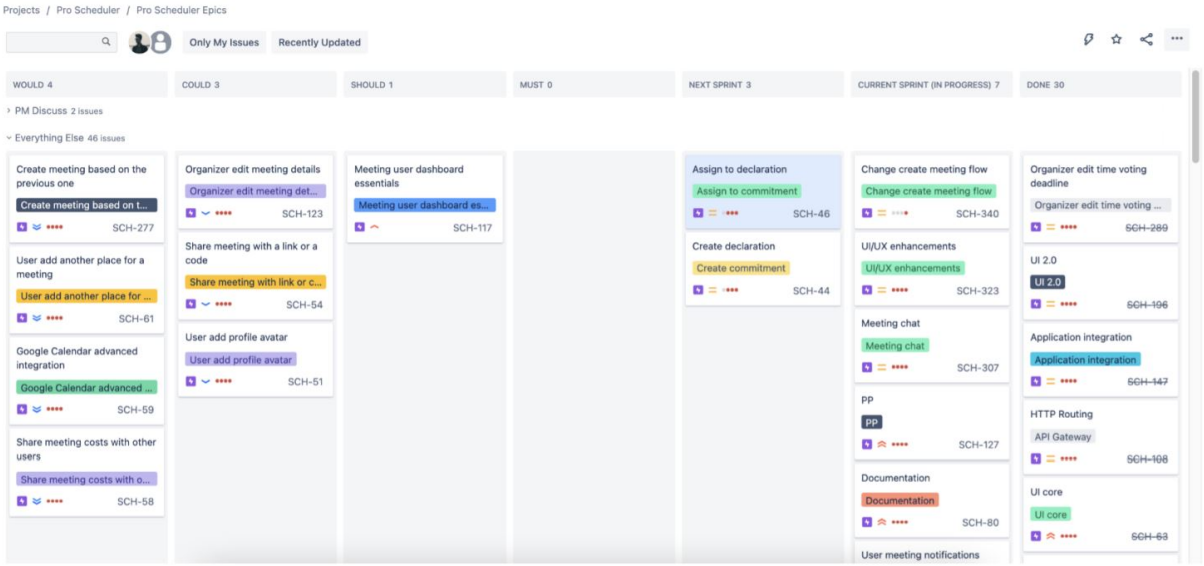

Figure 34: The board used to manage User Stories on the Jira platform

4.2.1.2. Scrum Poker

During internal meetings, as part of planning subsequent sprints, a web application was used to facilitate the allocation of points to individual tasks - Scrumpoker Online, available at `https://www.scrumpoker.online/`. Thanks to this tool, it was possible to determine the weight of each task in a very simple and intuitive way. An important advantage of using this solution was the fact that individual members of the team did not have the possibility to see other people's estimates, which occurs quite frequently when using standard voting communication channels. As a result, the distributed points very well reflected the real workload needed to complete a given task.

4.2.2. Source code

4.2.2.1. GitHub

The Git [16] technology and the GitHub [17] platform were used to store and manage the source code. In order to work effectively and maintain the code of all applications in the project, a GitHub organization was created, and then all team members were added to it.

The project consists of the following repositories:

- **proscheduler** - contains general information on how to prepare the entire environment for running, incl. scripts and configuration files for Docker, Nginx, or Jenkins tools

- **proscheduler-core** - contains the source code of internal libraries (Maven BOM, proscheduler IDS Spring Boot Starter) used by individual microservices

- **MeetingService** - source code of the microservice responsible for the creation and management of the meeting

- **SurveyService** - source code of the microservice responsible for the creation and management of surveys

- **DeclarationService** - source code of the microservice responsible for the creation and management of declarations

- **NotificationService** - source code of the microservice responsible for the creation and management of notifications

- **ChatService** - source code of the microservice responsible for the management of the chat in the meeting

- **WebApplication** - source code of front-end application

The GitHub platform was also used to implement the CI (Continuous Integration) process thanks to the offered GitHub Actions functionality. It allowed for the automatic building of artifacts and Docker container images of individual microservices, which were then placed on the DockerHub platform.

Other alternatives considered when selecting tools When selecting the tools at the beginning of the project, the following alternative tools, that are available on the market, were considered:

- **BitBucket** [6] - rejected due to the preferences of all team members

- **GitLab** [19] - rejected because it is mainly used for commercial purposes and because of the preferences of all team members

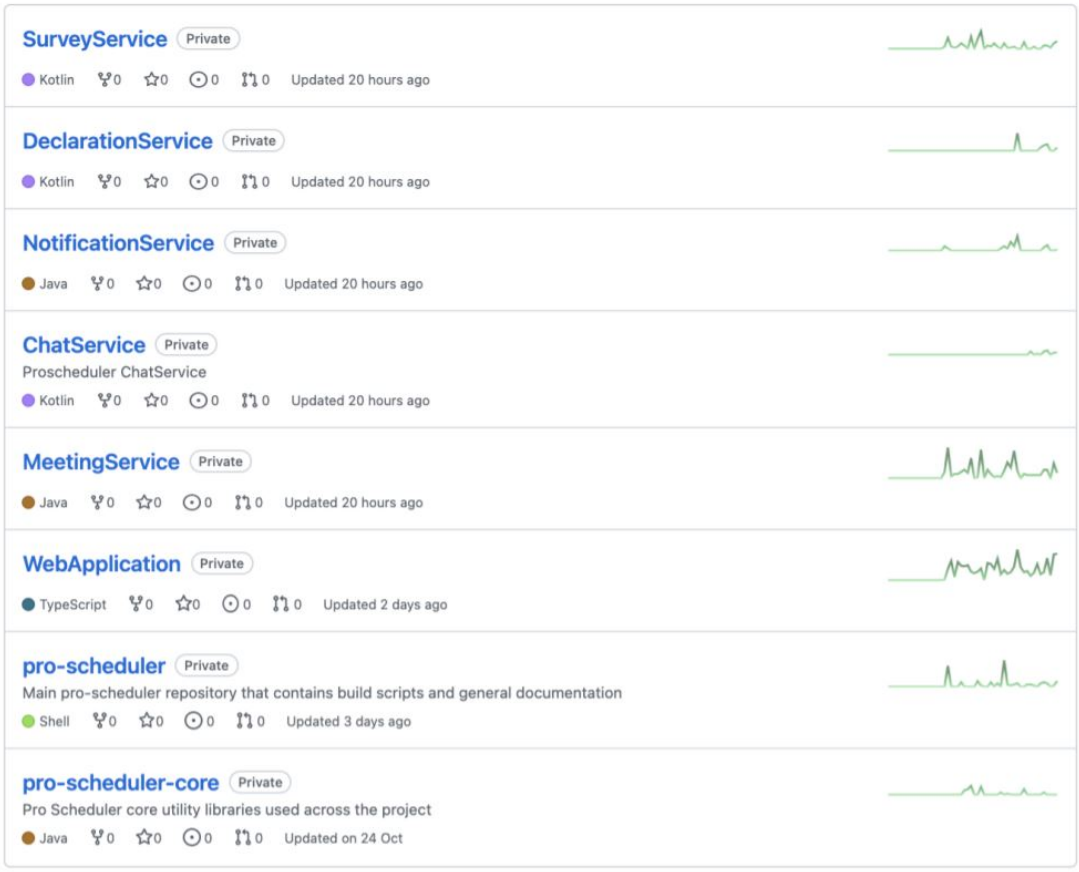

Figure 35: List of repositories on the GitHub platform with activity charts

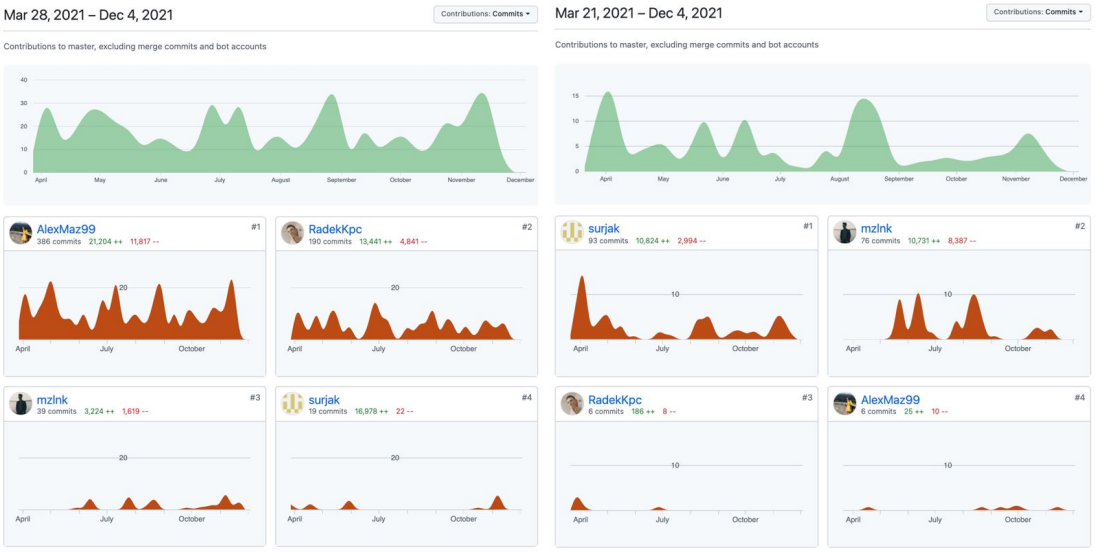

Figure 36: Sample graphs of the activity in WebApplication and MeetingService repositories

4.2.2.2. Artifactory

The Artifactory [5] platform created by JFrog was used for storing and versioning internal libraries and components used by individual microservices.

This platform was used to create a private Maven repository, to which subsequent versions of the above-mentioned libraries, created during the project implementation, were sent. This allowed avoiding problems related to the launch of individual websites locally by individual team members, such as:

- different versions of the installed library
- no given library installed in the local environment

4.2.3. Team communication

The main communication channel both in contact with the client and between individual team members was the Microsoft Teams [37] platform. For the purposes of the project, a dedicated channel was created on the platform, which allowed us to have the entire history of meetings and conversations always available in one place. On the platform, the following events took place:

- sprints planning
- meetings with the client
- pair programming
- discussions on the vision and implementation of selected project elements

Other alternatives considered when selecting tools When selecting tools for team communication at the beginning of the project, the following alternative tools, that are available on the market, were considered:

- **Discord** [10] - initially used in parallel with Microsoft Teams. However, due to poor video quality when sharing the screen with other members in real time was rejected in favor of the Microsoft platform

4.3. Practices and tools

During the implementation of individual tasks and goals from the project, a large number of techniques and good practices were used to support the work and maintain the quality of the produced software.

4.3.1. Mock-ups

One of the aspects of the project implementation, qualified as a risk during the process of designing the vision, was the demanding user interface. Therefore, we used AdobeXD software to create mock-ups that cover all views and components of the front-end application. Already in the first phase of the project (before starting the selection of target technologies and implementation), it was possible to consult the final appearance of the user interface with the client, which allowed to avoid possible misunderstandings regarding the planned goals. Of course,

due to the dynamic nature of the development process and creating new functionalities, some of the designed views evolved to adapt to the existing difficulties and ideas related to interface improvements (e.g. to increase the intuitiveness). A good example of this was the situation when we redesigned some components (tentatively called the process of creating UI 2.0 views) in order to adjust and standardize individual elements so that their styling and behavior were consistent with the current conventions and trends at the web application market.

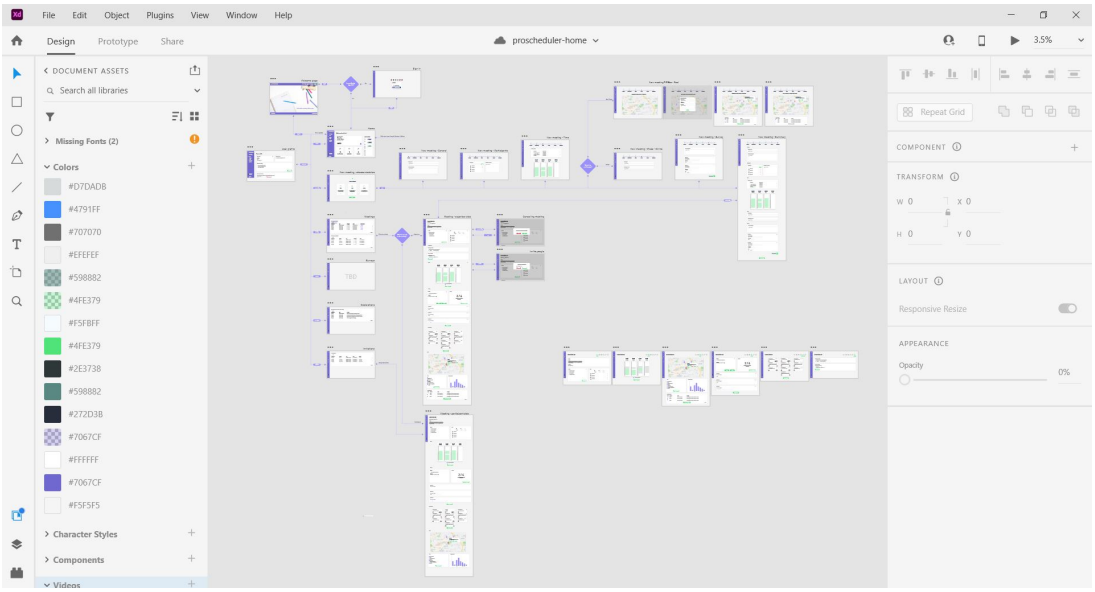

Figure 37: AdobeXD with a preview of all designed mockups for individual views in the application

4.3.2. Code review

Each newly implemented functionality, which was added by one team member, had to pass verification by other team members. Code review gives the possibility to maintain a high level of source code quality, as well as an additional opportunity for the rest of the team to familiarize themselves with the newly created implementation. This process was carried out using the functionalities offered by the Git tool and the GitHub platform, i.e. the branch model and the pull request system. This technique consisted in maintaining a stable version on the **master branch** in a Git repository and creating new changes on separate branches in accordance with the convention: **feature / <task code number on the Jira platform>**, e.g. SCH-154. These changes could then be added to the existing code via a pull request, in which the code review was performed by other team members, and unit tests were automatically run.

4.3.3. Code refactoring

Due to the developmental nature of the project and the evolution of some solutions during the implementation of tasks, it was very important to track the increasing technical debt. Therefore, the most important elements of the product were refactored several times, both on the back-end and front-end. This refactoring mainly involved the unification of the architecture within a given component or shared API interface.

4.3.4. Unit tests

A part of the developed software on the server-side was covered with unit tests in order to verify the correctness of the implementation. Due to the remarkably dynamic growth of new functionalities imposed by the set of goals, only the most important and critical part of the code was covered with the above-mentioned tests to make sure that a given implementation (often used by other components via API) works correctly. This approach makes it much easier to debug the product and find places where a given error occurs.

4.3.5. Pair programming

During the implementation of some tasks (in particular those related to the implementation of components) the pair programming technique was often used due to the numerous benefits. One of them is a faster implementation of a given issue, especially in very complex components. Parallel creation of the software code gave us the possibility of mutual consultation of the ideas of both parties. This technique turned out to be very useful in creating initial prototypes of subsequent microservices (Proof of Concept) as it verified very clearly whether our concepts were possible or not.

5. Project results

The final result of the project is a fully operational and functional application, available for public use at `https://proscheduler.io`.

To confirm the usability of our application and the quality of the offered user interface, we conducted a validation process with real users, which is devoted to the next section.

5.1. User Experience

In the period from November 27, 2021, to December 3, 2021, UX tests were carried out, enabling feedback gathering from users using our application. An anonymous questionnaire with the use of 'Google Forms' was created for this purpose, consisting of a set of preliminary questions, allowing for the collection of basic information about the participants of the study, and four tasks requiring getting acquainted with the tested product. At the end of each challenge, there were questions about the degree of its implementation, difficulty level, problems encountered, and suggested changes to the application.

5.1.1. Research group

The UX tests included 20 people aged from 17 to 40 who spent at least a few hours a day on the Internet, three-quarters of them indicating that they are related to the IT industry. People who might be interested in using our application in the future were selected to participate. The interviewees used a variety of devices during the tests, such as laptops, desktops, tablets, and mobile devices, as well as many web browsers, with Google Chrome and Mozilla Firefox being the most popular. The statistics on their use are presented below.

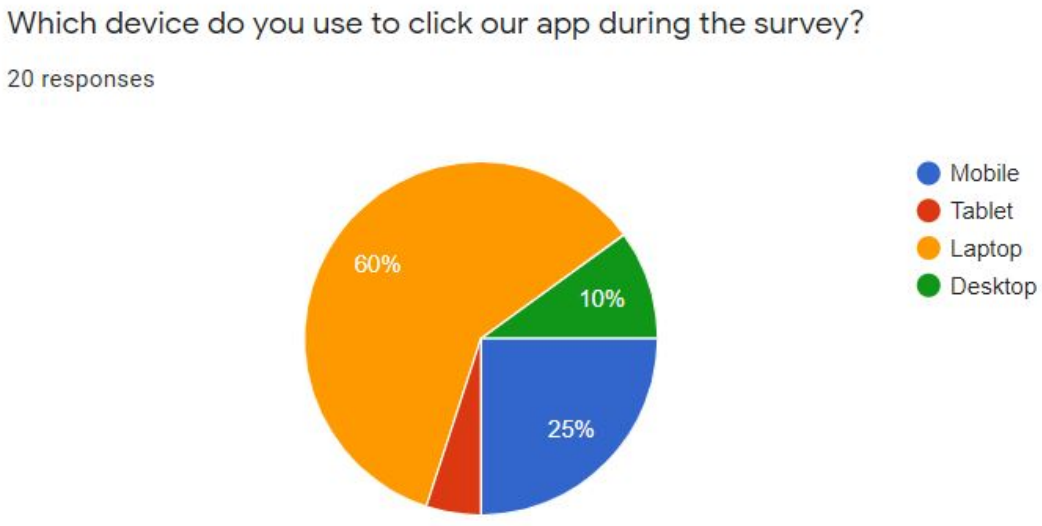

Figure 38: Devices used by users during the UX tests

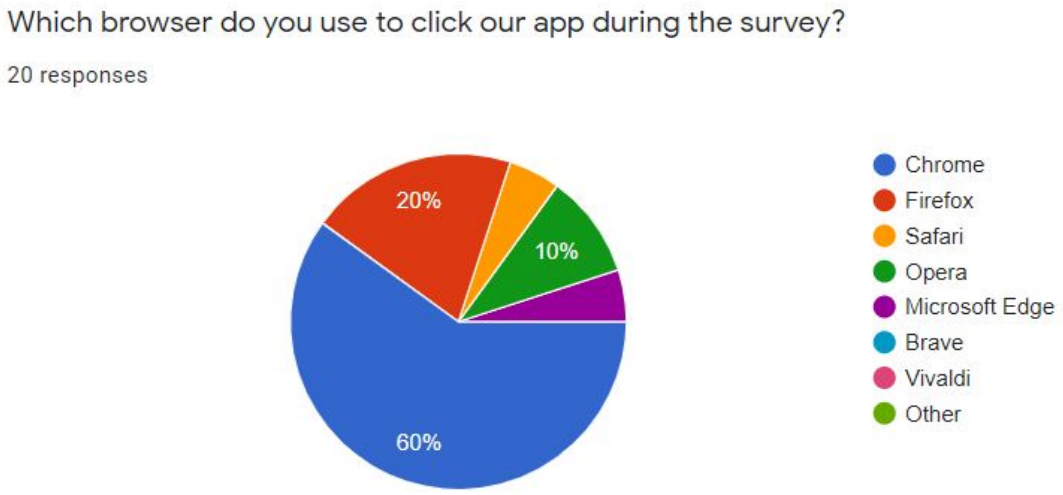

Figure 39: Web browsers used by users during the UX tests

5.1.2. Challenge 1

The first challenge was to log in to the application using the indicated link, and to arrange a meeting with friends at a local cafe then. For this purpose, the users were asked to create a real meeting taking into account all available modules, invite a participant with the given email address, select many possible time slots and potential places, and create a survey with two questions of different types.

5.1.2.1. Challenge results

The results of the first task were satisfactory as most of the users accomplished it successfully and only two of them failed to complete it properly. The challenge was rated as fairly easy, with a score of 8.8 points on a possible scale from 0 (difficult) to 10 (easy). Detailed results are presented below.

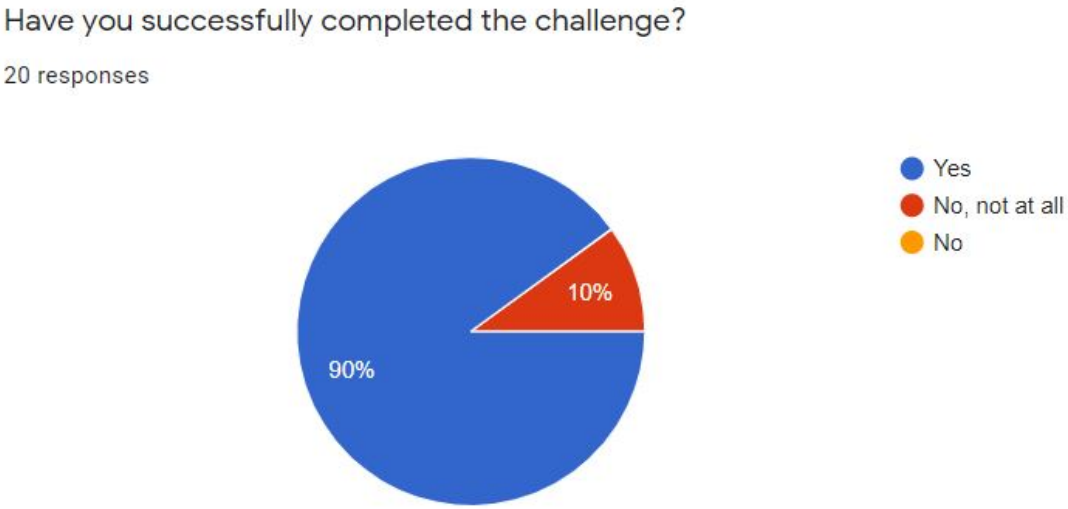

Figure 40: Degree of completion of the first challenge by users

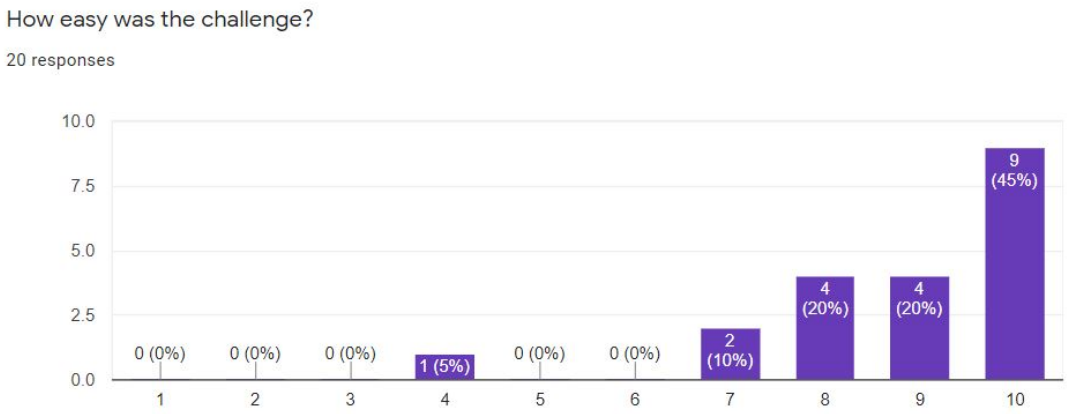

Figure 41: Difficulty level of the first task (0 - difficult, 10 - easy)

5.1.2.2. Comments and introduced changes

Users made some comments and suggestions about using the application that they noticed during the first task. Due to the fact that we care about the satisfaction of people using our product, we tried to include all of them in the final version of the project.

The first of the comments received was the non-intuitive switching between the stages of creating the meeting. Initially, to go to the next module, you had to click the arrow next to the navigation bar at the top of the screen. Some users reported that they had a problem with this in the beginning, so an additional button at the bottom of the screen was added to the existing navigation. Both its placement and content should increase the intuitiveness of the meeting creation process.

After analyzing the remaining suggestions, we have added to the home page a button redirecting to the meeting creation view in order to encourage the user to take an action right after logging in.

5.1.3. Challenge 2

In the second challenge, the interviewees were asked to change the username to the name of any animal.

5.1.3.1. Challenge results

The results of the second task were fully satisfactory as all users completed it successfully and rated it as very easy, giving it 9.7 points, where 0 means difficult and 10 - easy. Detailed results are presented below.

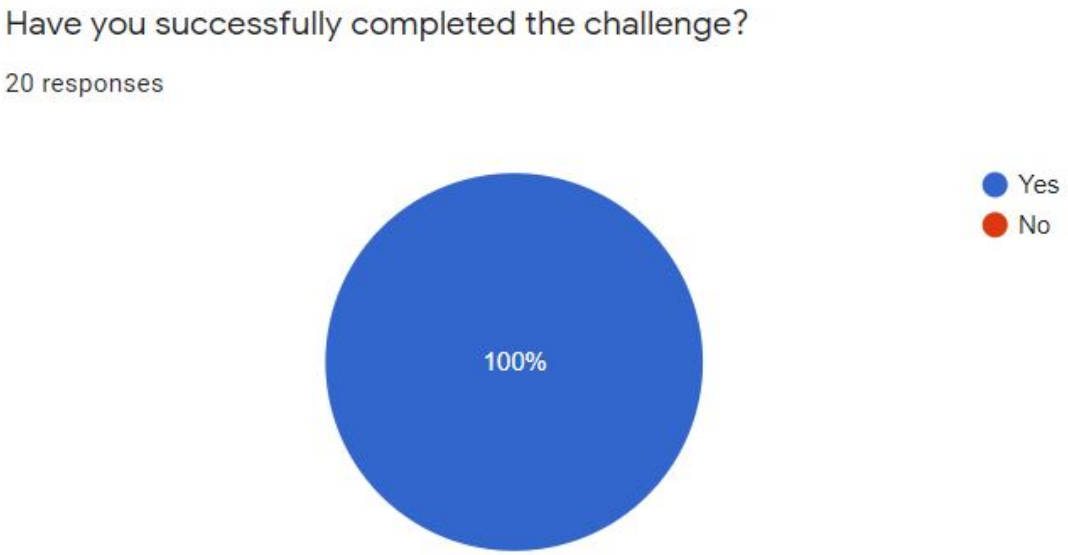

Figure 42: Degree of completion of the second challenge by users

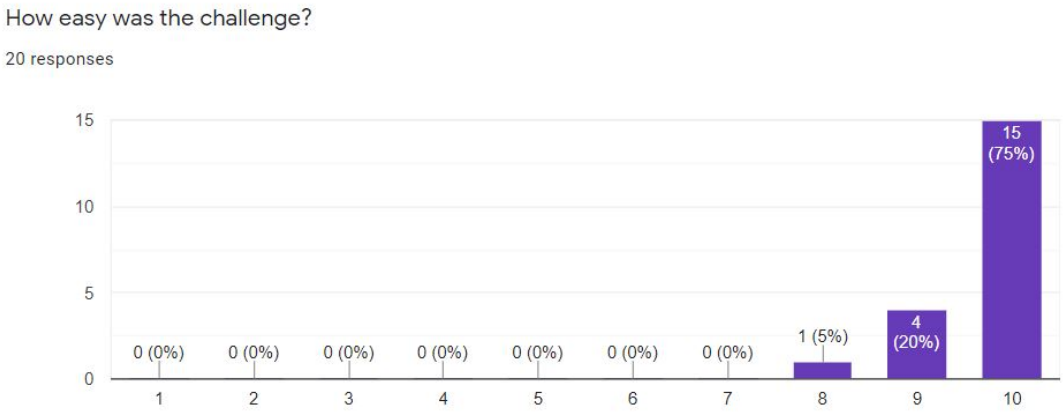

Figure 43: Difficulty level of the second task (0 - difficult, 10 - easy)

5.1.3.2. Comments and introduced changes

The only remark raised by one respondent was the observation of an undesirable - in their opinion - effect occurring after clicking the application's logo, which was redirecting the currently logged-in user to the landing page. Due to the fact that several users also found this effect irritating in the further part of the survey, we decided to change the sequence of clicking on the logo to being redirected to the home page.

5.1.4. Challenge 3

In the third challenge, the users were asked to find the meeting created in the first challenge and set its final date and place then.

5.1.4.1. Challenge results

The third task can be considered as the most difficult one in the entire test, due to the fact that one person failed to complete it and the average difficulty level was 8.55 with a possible scale from 0 (difficult) to 10 (easy). Detailed results are presented below.

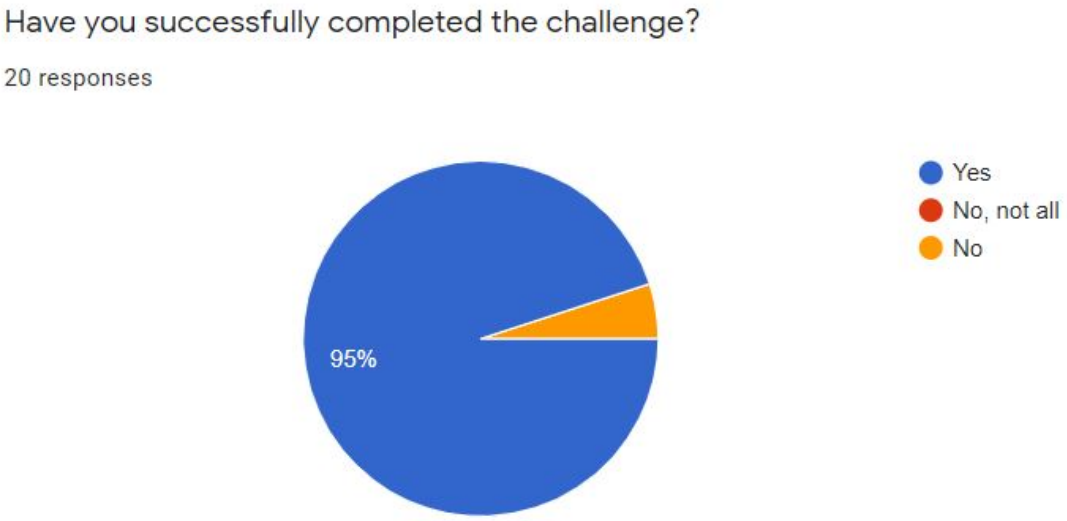

Figure 44: Degree of completion of the third challenge by users

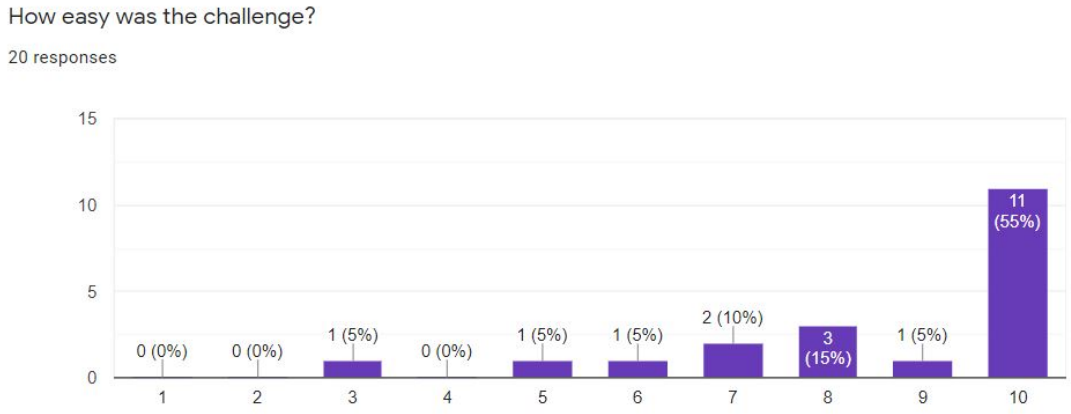

Figure 45: Difficulty level of the third task (0 - difficult, 10 - easy)

5.1.4.2. Comments and introduced changes

The respondents stated that they had no problems finding the previously created meeting or choosing its final place. However, not everyone managed to set its final date which can be a result of the inconsistency between these activities, as the choice of the venue could be made both in the general section of the meeting and in the voting tab for the final place, and setting the time was possible only in the general section. Due to the aforementioned comments and in order to make the application more consistent, the option to select the final date of the meeting has also been added in the time voting tab.

Furthermore, one user commented that there was the presence of two different time formats in the views. In order to standardize them, the AM / PM system was abandoned in favor of the 24h system and appropriate changes were made to all components.

5.1.5. Challenge 4

In the fourth challenge, the respondents were asked to join the meeting via provided link and complete the following tasks: select time preferences, vote for any place, complete the survey, create and assign to a selected declaration.

5.1.5.1. Challenge results

The performance of this task was satisfactory as almost all users successfully completed the challenge and rated it as a simple one, giving it 9.6 points, where 0 is difficult and 10 is easy. Only one of them failed to fully complete the task. Detailed results are presented below.

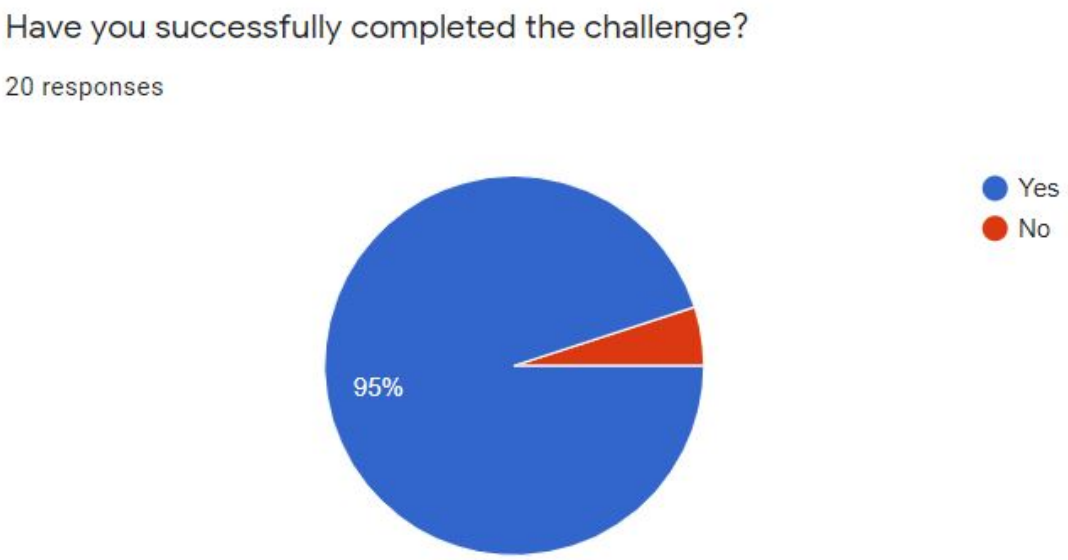

Figure 46: Degree of completion of the fourth challenge by users

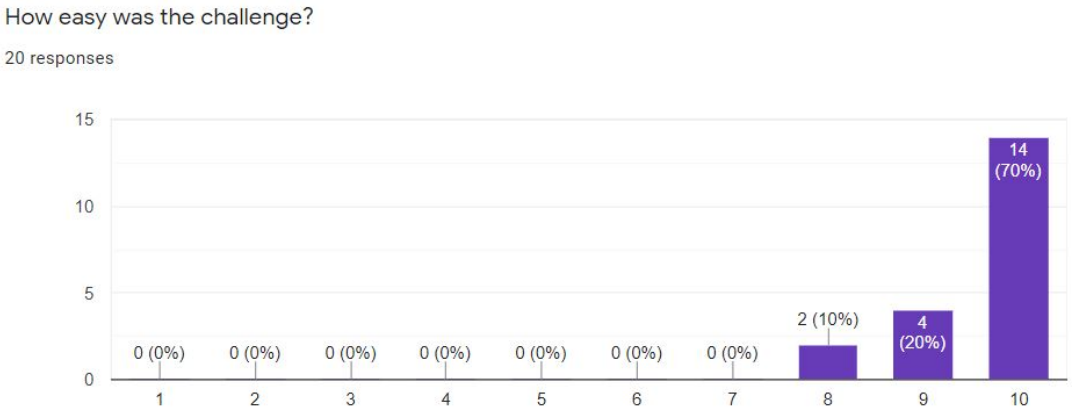

Figure 47: Difficulty level of the fourth task (0 - difficult, 10 - easy)

5.1.5.2. Comments and introduced changes

Users made some comments and suggestions about using the app that they noticed during the last challenge.

The first one was the non-intuitive nature of the switch button, which changed the views of a given user's votes and the current voting results. Therefore, the aforementioned button has been replaced with two buttons with appropriate captions.

The second remark was the lack of responsiveness of the navigation bar used to move between different modules in the meeting details view. As a result, on mobile devices, the button redirecting to the declaration was not visible. This consideration was taken into account by replacing the aforementioned navigation with a dropdown button that appears when using the application on smaller screens.

5.1.6. Summary

Finally, the respondents were asked to write what they like the most about our application and what they do not. The answers received are presented below.

Strengths:

- offered functionalities
  - possibility of selecting continuous time intervals
  - declarations module
  - surveys module
  - ability to mark places on the map
  - possibility to add a meeting to Google Calendar
- the appearance of the application
  - clean and intuitive design
  - graphs animations in the voting results preview
- degree of product development
- product performance
- ease of use

Weakness:

- non-intuitive movement between modules in the meeting creation view (improved)
- clicking the application's logo takes you to the landing page (changed)
- AM/PM and 24h time formats (changed)

5.2. Final version of the application

This chapter presents the final version of the application with a detailed description of each view. The summary in each section will focus both on the appearance and functionalities offered by a given subpage. In line with the client's expectations, we have managed to achieve a consistent and clear user interface. The most important is the fact that all the most significant requirements have been implemented, as well as a number of additional functionalities (e.g. integration with Google Calendar).

5.2.1. Login page (landing page)

Login page view is the initial page of our application which the user is directed to after accessing the application at proscheduler.io. It contains basic information about the application including available functionalities and also indicates the possibility of logging in using a Google or Facebook account. From the business point of view, this page is very important because it makes the first impression on the user and it is intended to encourage the potential recipient to take advantage of the product. While creating this view, we wanted to focus on simplicity and convenience, therefore the login buttons are placed in a visible place, and the inscriptions promoting our application are subtly eye-catching to gather user's attention.

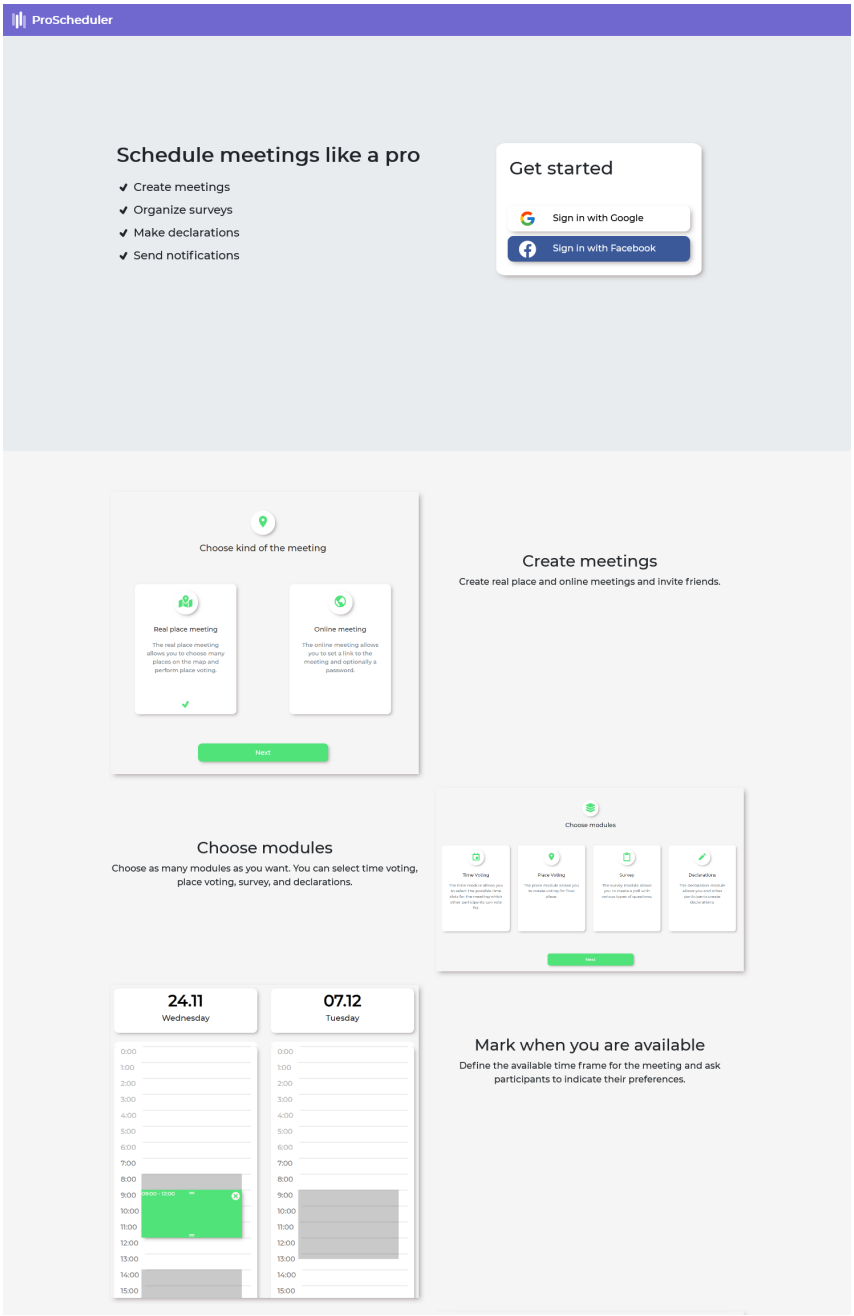

Figure 48: Landing Page - part 1

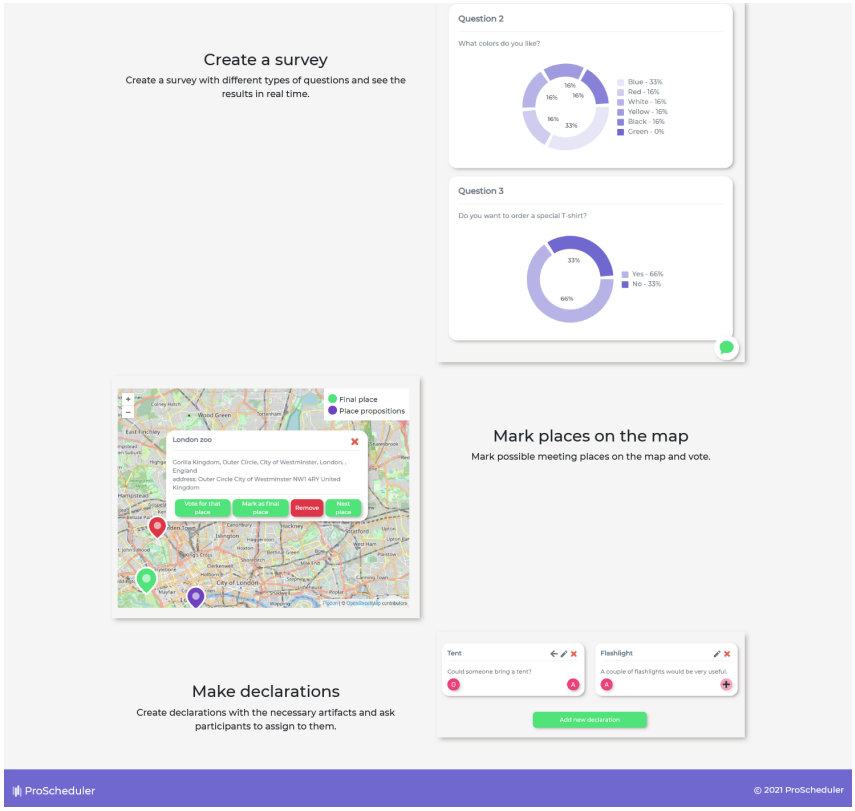

Figure 49: Landing Page - part 2

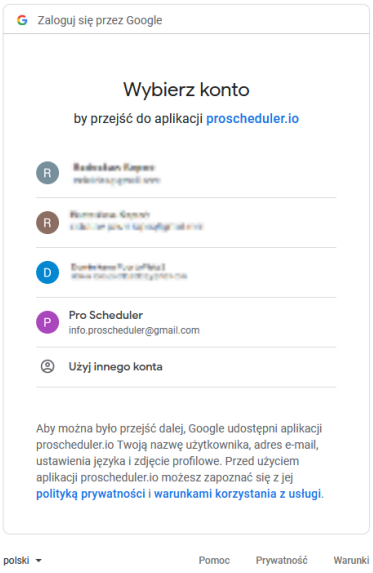

Figure 50: Signing in with a Google account

5.2.2. Home page and navigation

Home page is the first page the user sees after successfully logging in. It contains the most important information for the user like upcoming meetings, personal calendar for current day, amount of new invitations, amount of surveys to fill or created declarations. The page is also

accessible directly from the navigation bar located on the left of the screen - the user has to click on the application logo to navigate to it. The home page consists of plenty of “quick access” buttons to make users easily access other parts of the application in a few steps. The aforementioned navigation bar consists of the following elements:

- application logo - redirects to the Home Page
- “New Meeting” - highlighted section which allows proceeding to meeting creator wizard
- “Meetings” - list of user’s current and past meetings
- “Surveys” - list of user’s surveys to fill
- “Declarations” - list of user’s created declarations
- “Invitations” - list of current meeting invitations
- “Hide” - utility button to expand or collapse the navigation bar
- user profile icon - user’s profile and settings
- log out icon

The expanded navigation bar and the Home Page view for desktop and mobile devices are presented below.

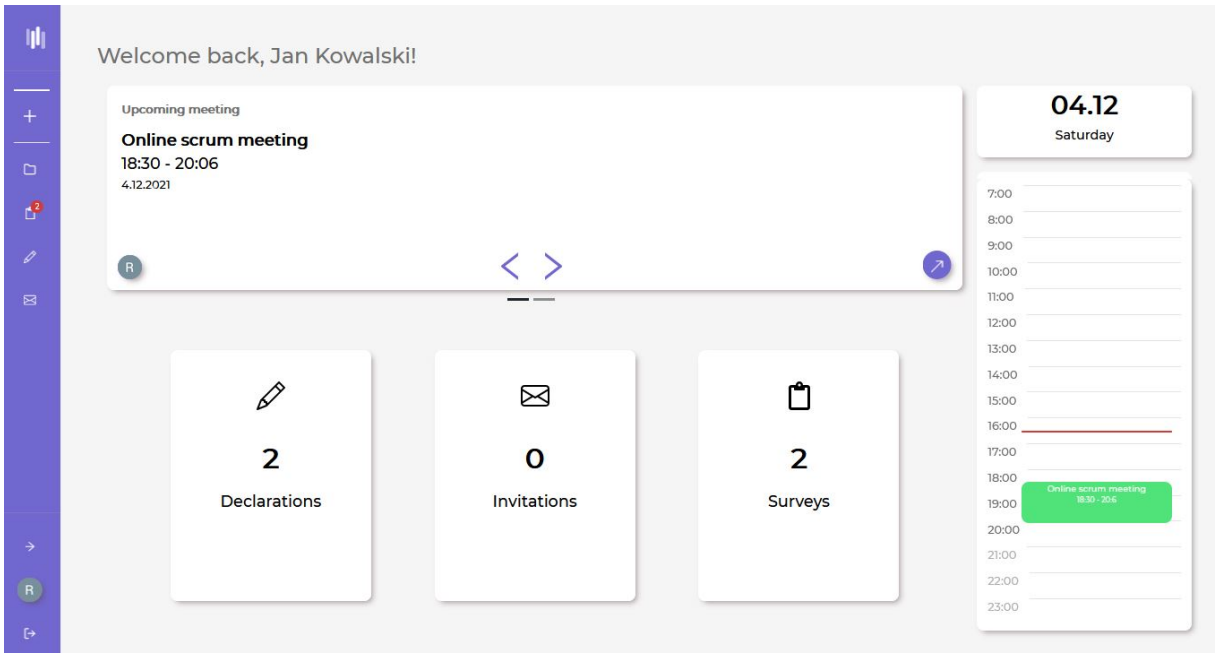

Figure 51: Home Page view for desktop devices

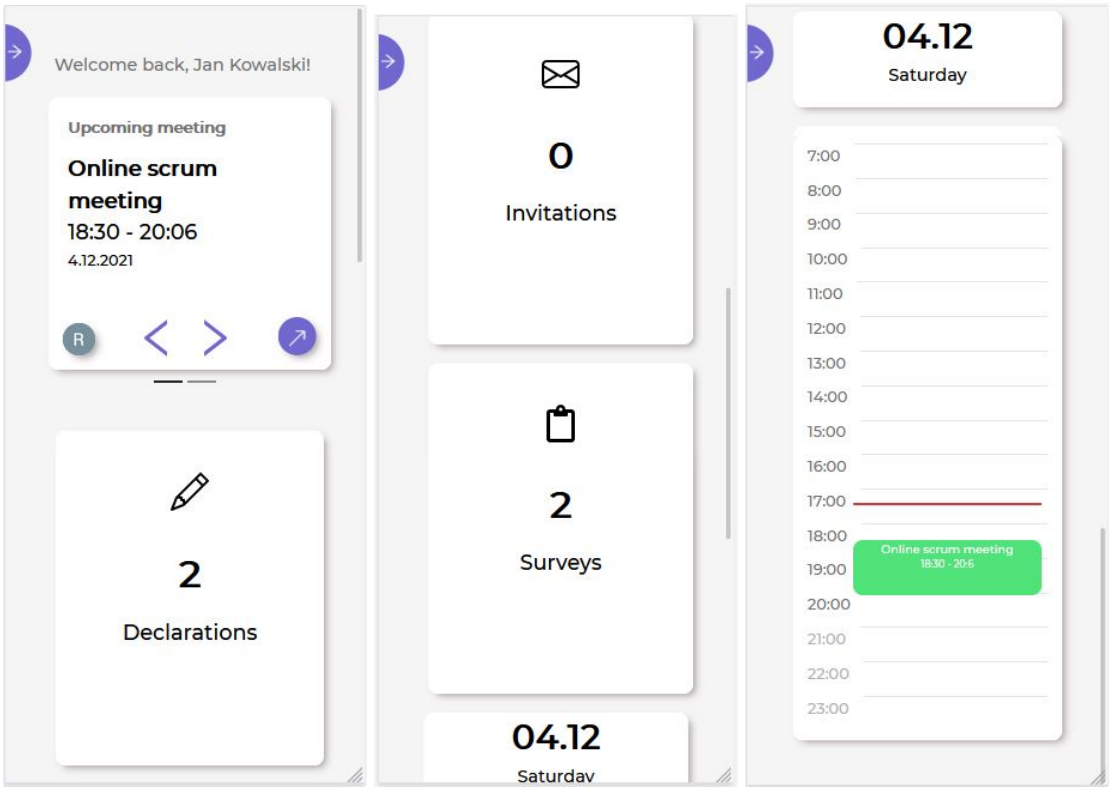

Figure 52: Home Page view for mobile devices (IPhone X/XS iOS 12)

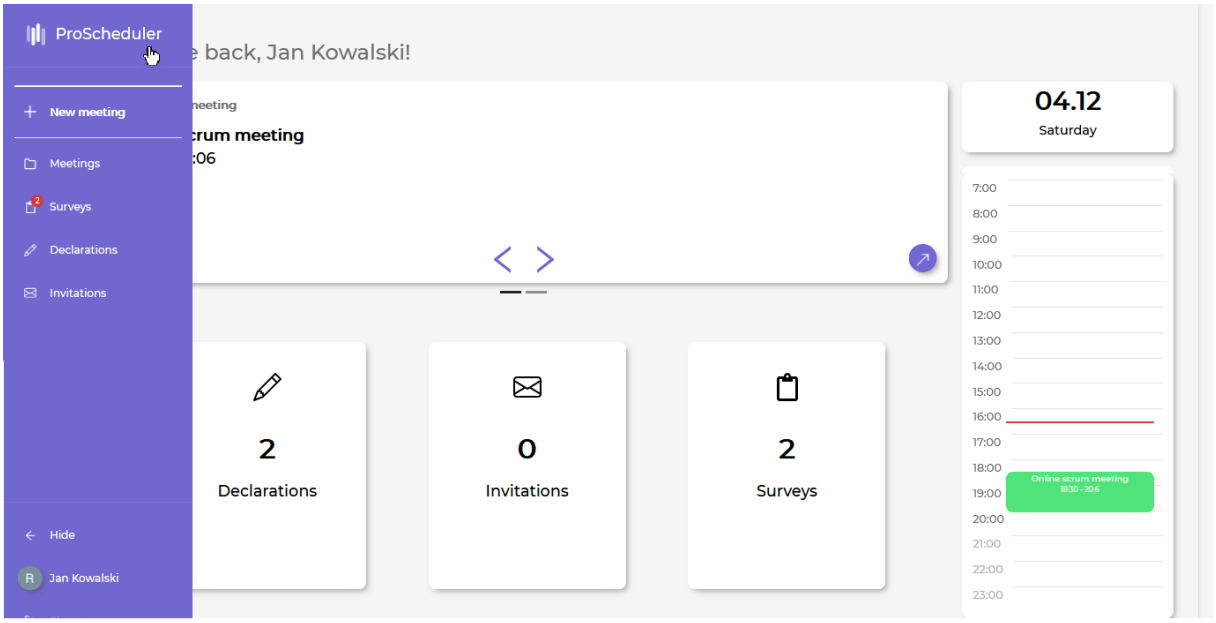

Figure 53: Navigation bar for desktop devices

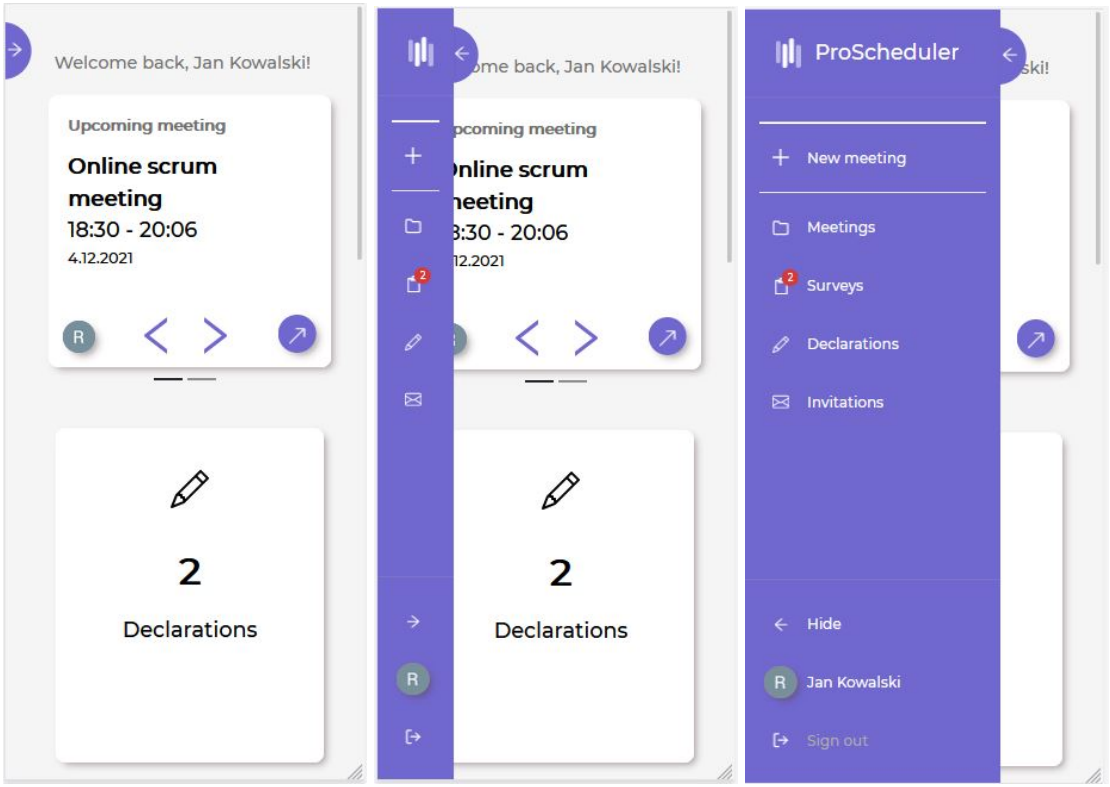

Figure 54: Navigation bar for mobile devices (IPhone X/XS iOS 12)

5.2.3. User profile view

The User Profile view consists of essential user settings and it allows users to customize several parts of the application which makes this view really important regarding the user experience aspect. In the “User Details” tab users can change their visible name in the system (the default visible name is an email address). In the “Notification settings” tab users can choose whether they want to receive automatic notifications related to the meeting surveys or time voting deadlines. They can also configure other notification settings such as accepting messages from meeting organizers or receiving invitations via email. The user has also the possibility to remove their account via the option located at the “Remove your account” tab. This functionality is crucial because it makes our application in line with current guidelines related to GDPR rules. The last tab called “Change your theme” is related to visual customization of the app. The user can choose their own application color palette and such changes will be applied throughout the whole application.

The User Profile view for desktop devices and the result of personal theme customization are presented below.

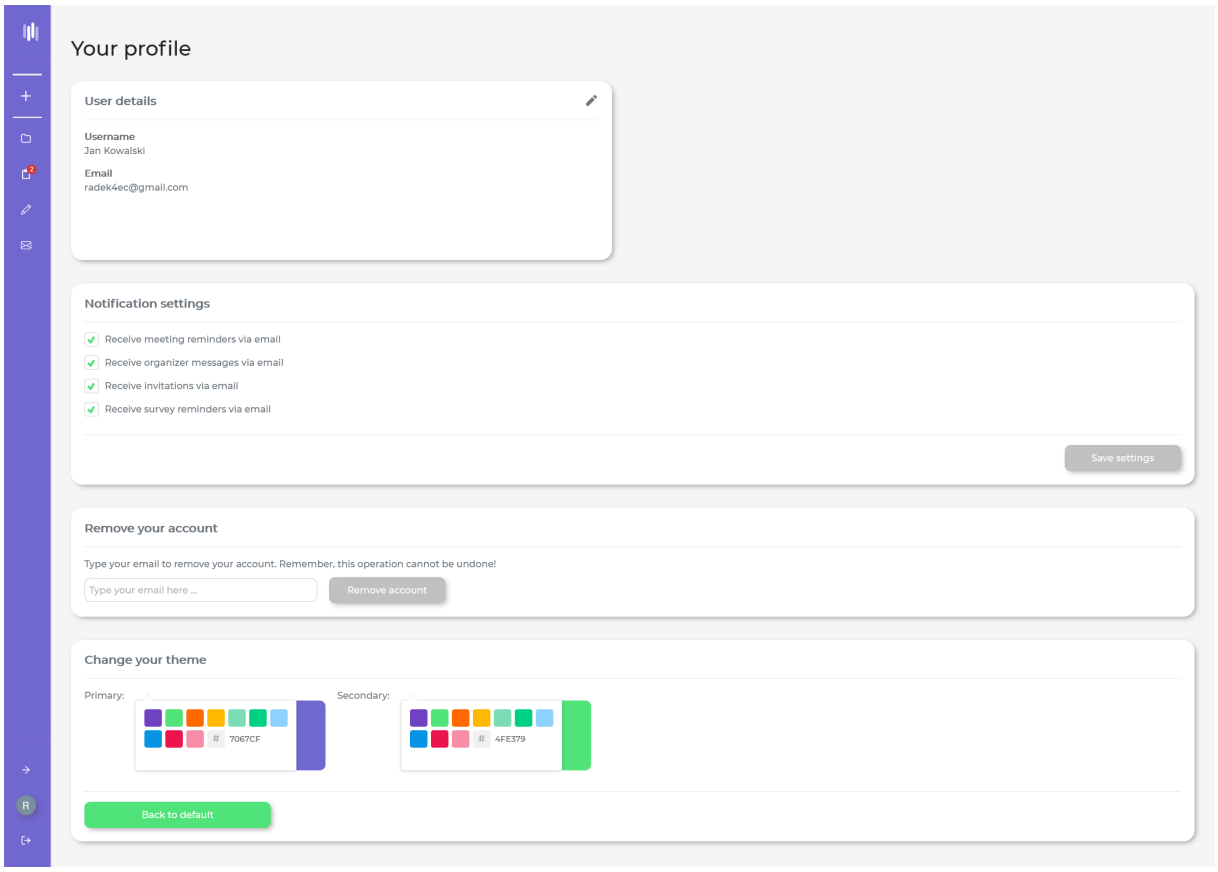

Figure 55: User Profile view for desktop devices

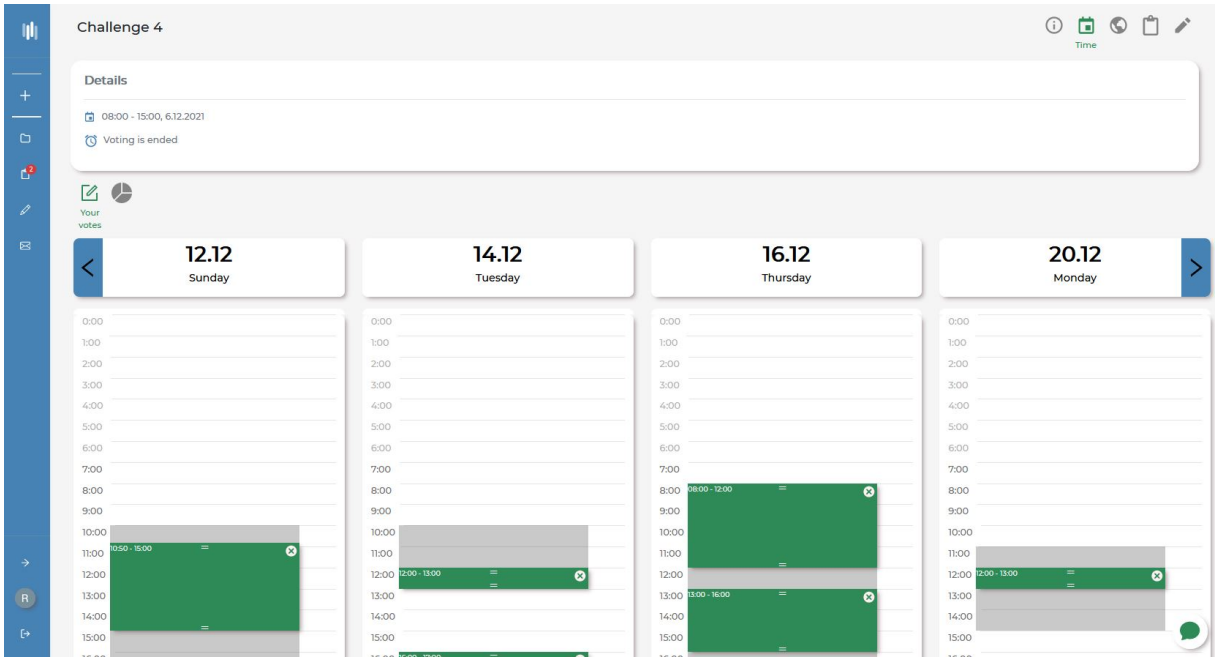

Figure 56: The result of personal theme customization

5.2.4. Meeting creation view

The meeting creation view is one of the two most important views in the application as it allows a user to create a new meeting which is the core functional object of the product. One of the biggest challenges we have faced was to provide a simple and intuitive interface that presents particular stages of creating the meeting. We decided to make this view in the form of a creation wizard where a user can configure a meeting step by step providing necessary information based on previously chosen options.

In the first step, the user has the option to choose the type of meeting - real or online. For the first type of meeting, the user has the possibility to mark the exact location for a possible meeting place in further steps. On the other hand - if the online meeting type is chosen - there is an option to provide a link and optional password for the meeting site.

The following figure presents the first step of creating a new meeting where the user chooses between different types of meeting.

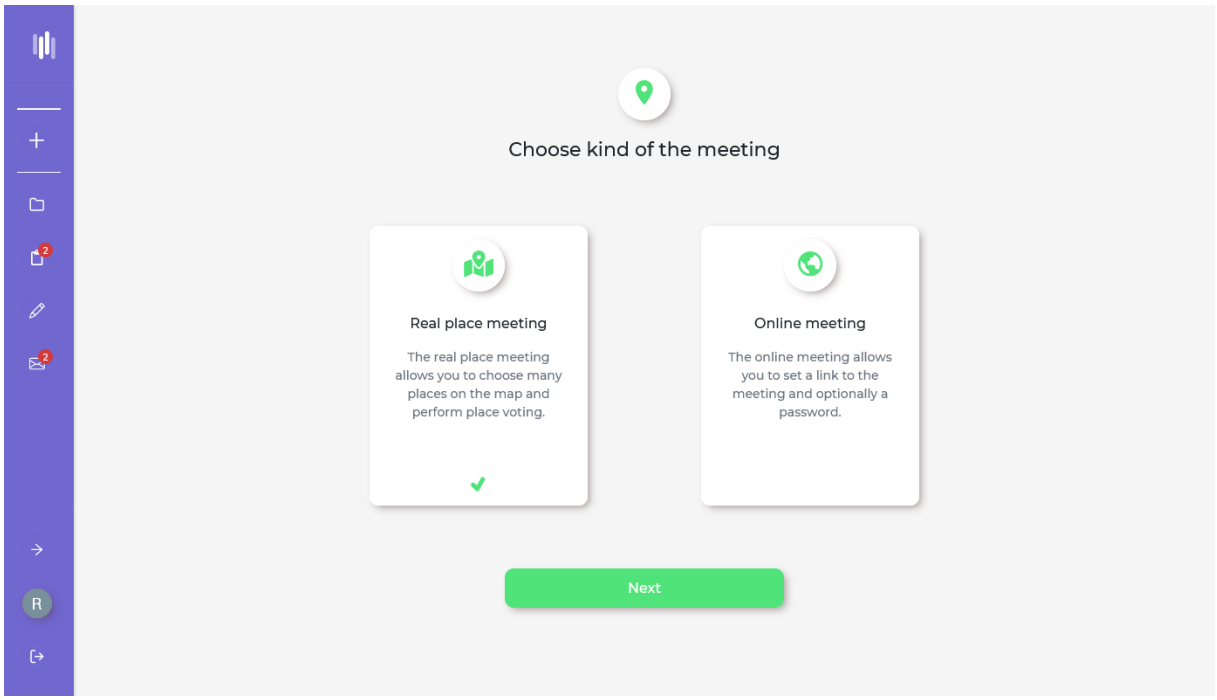

Figure 57: Meeting creation view - choosing meeting type

When the user successfully chooses the desired meeting type and moves forward - they will be navigated to the section where they can choose multiple additional modules (module is a set of functionalities for meeting management). In the current implementation of the product there are available four different modules:

- Time Voting - allows meeting participants to provide the organizer their time preferences for the meeting
- Place Voting - allows meeting participants to vote for meeting place
- Survey - allows meeting organizer create survey related to the meeting
- Declarations - allows meeting participants to create and assign for declarations (e.g. “Someone needs to take extra sleeping bag”)

The figure below presents a view where a user can choose different modules.

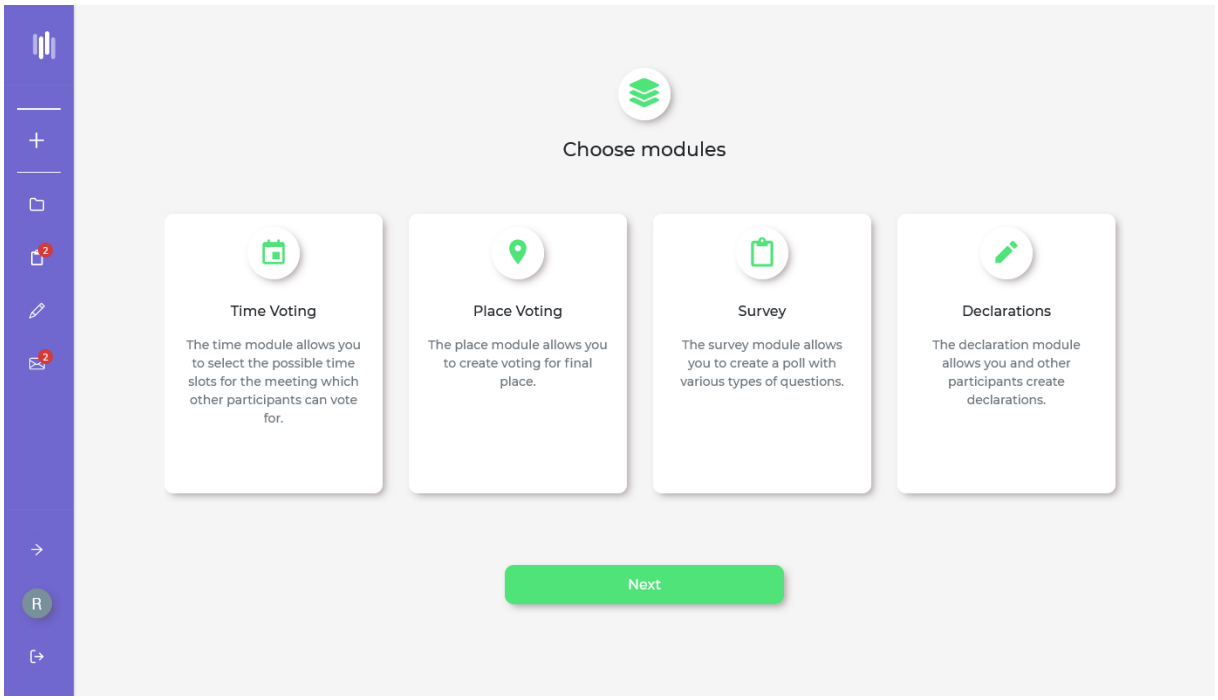

Figure 58: Meeting creation view - choosing meeting modules

In the next step - when the user chooses modules based on their needs - they are navigated to the view where they have to provide necessary meeting details. First of all, the user needs to specify a mandatory meeting name and an optional description (Figure 29.3). At the very top of the view, there is a bar representing the progress of creating an appointment and the current step is visible. You can navigate through the wizard using the arrows at the top navigation bar or using the ”Next” button at the bottom of the screen.

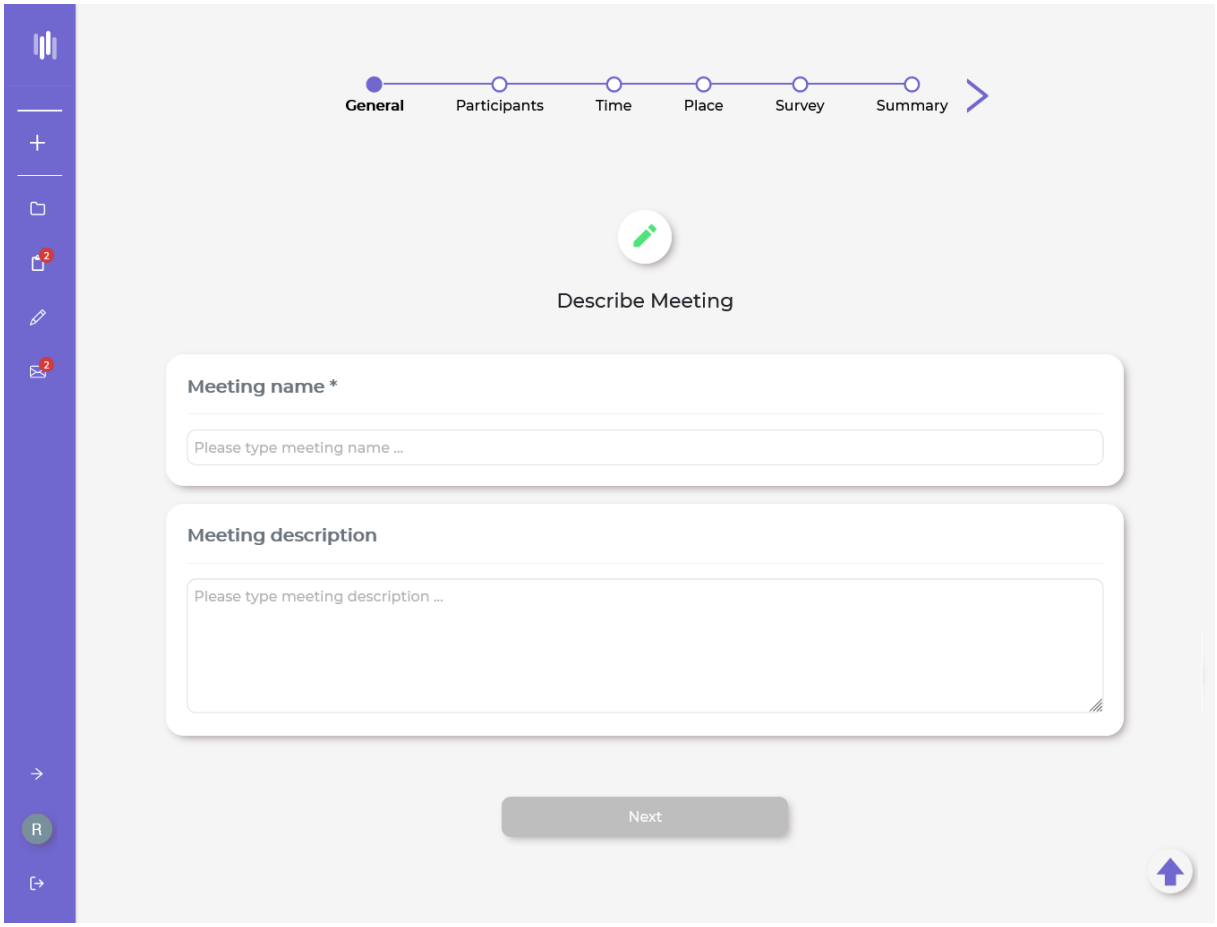

Figure 59: Meeting creation view - specifying meeting details

After completing the first step, we move on to inviting participants. In this step, we can add meeting participants via their emails and we can specify the message they will receive after sending the invitation. This is an optional step as we can invite participants later either directly or using the invitation link.

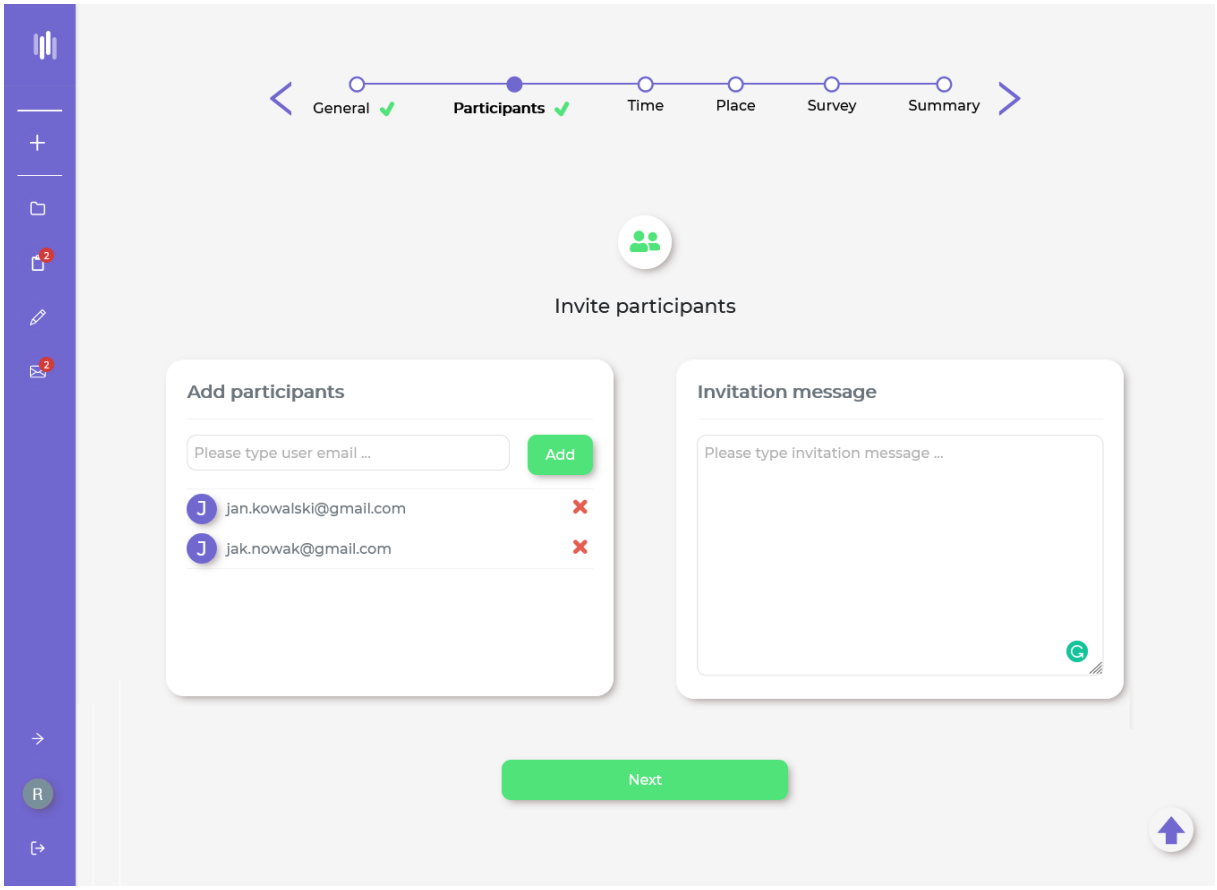

Figure 60: Meeting creation view - inviting attendees

The next step (if we have chosen the time voting module) is to select the days and times for the potential meeting time (participants will be able to select their time preferences in these intervals). The days are selected by using the calendar. The choosing of time periods is made using our custom component, described in detail in chapter 3.8.2.1. Figure below presents the described above module.

Figure 61: Meeting creation view - selection of time intervals

The next step (if we have chosen the voting place module) is to mark the places selected for voting on the map. You can use a search bar that uses geocoding to find a suitable place after the entered phrase, or you can click directly on the map. Adding a place to the map will automatically download information about it from the geocoding service. It is very useful because the user does not have to complete the address of the indicated place. After adding a place, a modal appears where you can change or fill the data about the selected place. It is worth mentioning that the organizer does not have to select all places immediately, you can add, delete and edit them later.

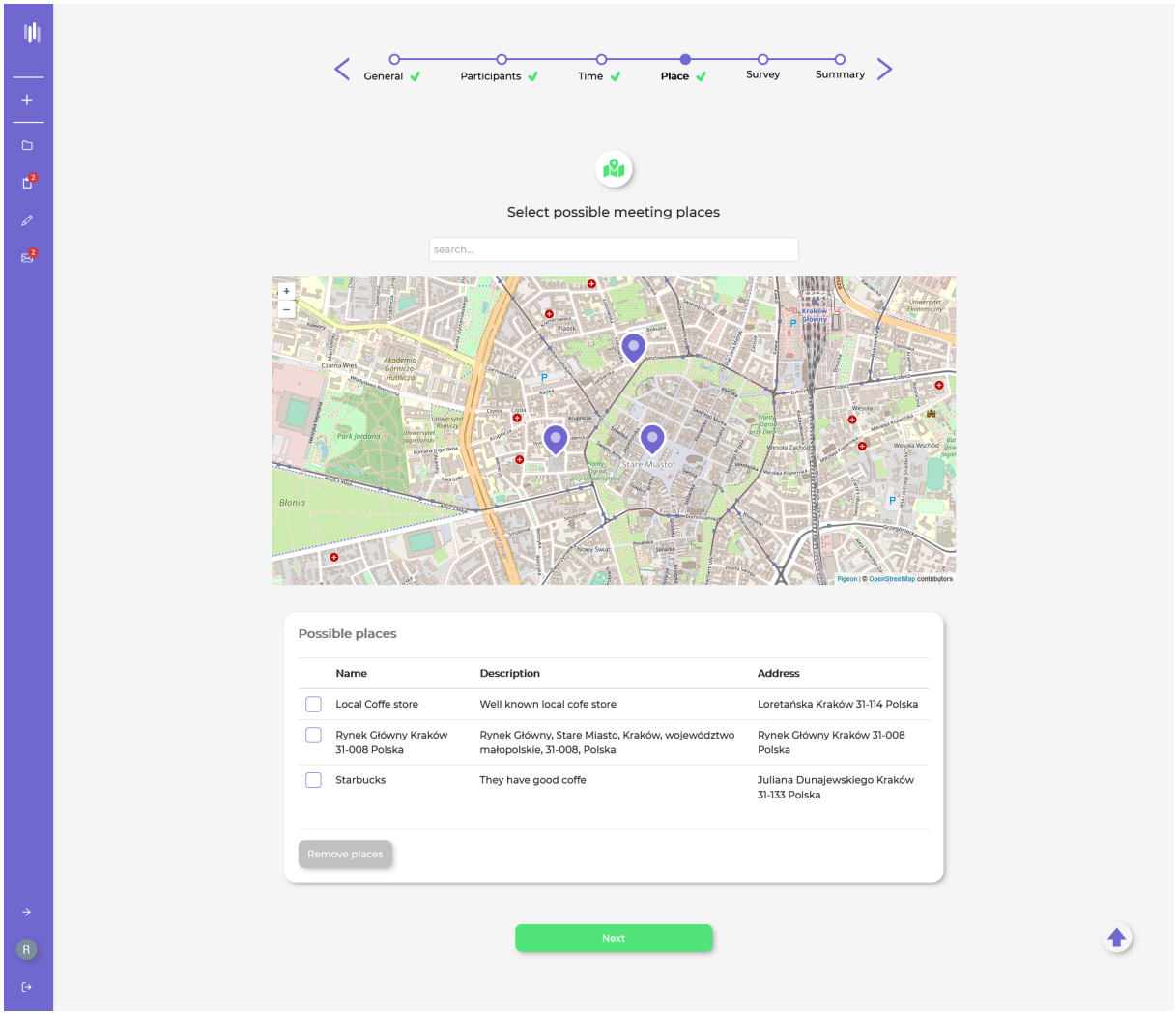

Figure 62: Meeting creation view - choosing places to vote

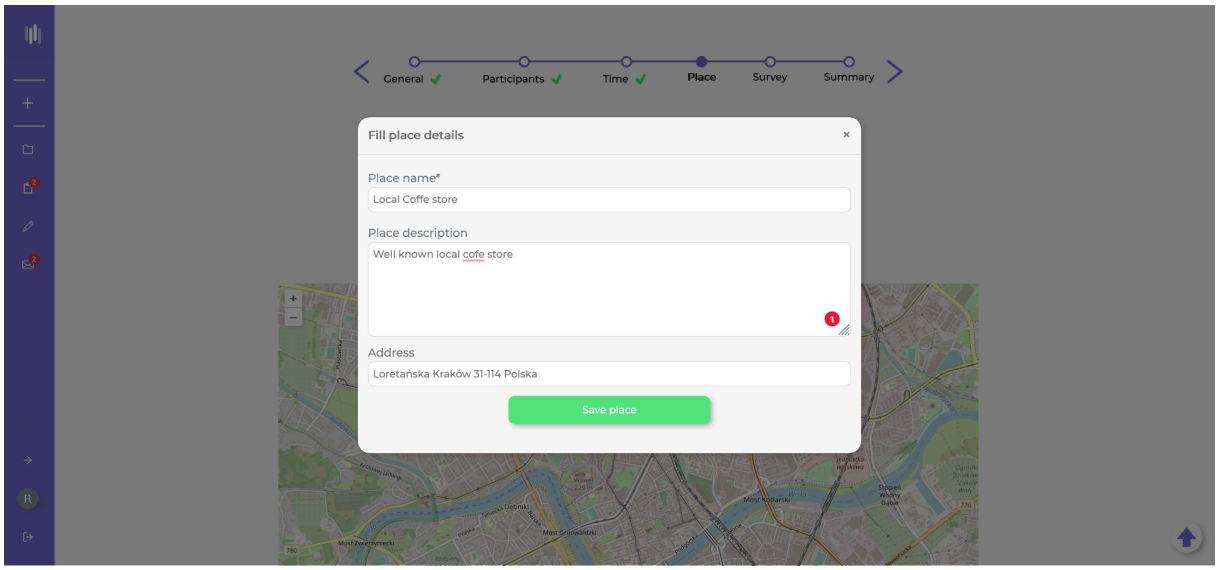

Figure 63: Meeting creation view - specifying details of the place

After selecting the places, we can add various types of questions to our survey (if we have previously selected the survey module). The question pool can later be edited by the organizer. We specify a description of the survey and use the "Add question" button to add some questions. We can choose from five different types of questions:

- open
- closed with single choice
- closed with multiple choice
- yes or no
- with a scale of e.g. 1 to 10

Below you can see the creating survey view in which all types of questions have been placed.

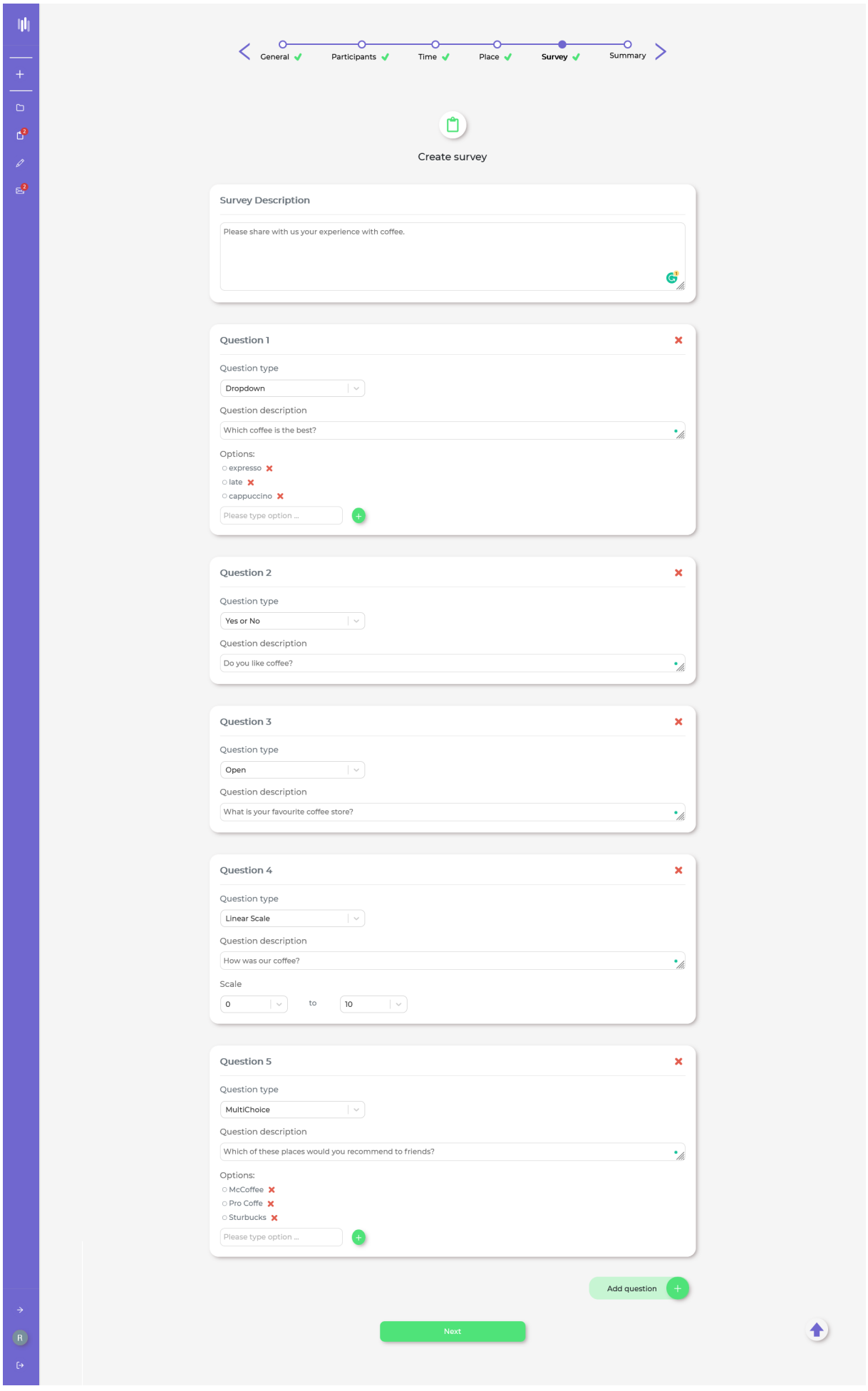

Figure 64: Meeting creation view - creating a survey

The last step is summary, where you can review the activities performed in the previous steps, and finally create a meeting by clicking a submit button at the bottom of the view.

5.2.5. Meeting details view

The meeting details view is the second of the two most important views, as it contains most of the key functionalities of the application. The view is divided into a maximum of the six following sections, the number of which depends on the selected modules.

- “About” - section with basic information about the meeting
- “Time” - section with time voting
- “Place” - section with place voting
- “Survey” - section with a survey
- “Declarations” - section with declarations
- “Settings” - section with meeting settings (visible only to organizers)

The navigation between the selected sections is possible through the corresponding icons, located in the upper-right part of the screen. Importantly, certain view elements, such as editing, settings, or voting results, are by default invisible to participants who do not act as an organizer. A significant element is also the meeting chat through which participants can communicate by exchanging messages. All the available sections are presented below.

5.2.5.1. Meeting details view - section “About”

The section “About” in the meeting details view is the first section shown to the user after entering the view. It consists of some basic information like meeting name, description, final place and time and also meeting participants. The meeting organizer can manage the meeting in this view with following actions:

- edit meeting name and description
- make another participant a meeting organizer
- choose a final place and date for the meeting
- invite a person to the meeting
- remove the participant from the meeting
- generate an invitation link
- close voting
- leave the meeting
- cancel the meeting

Additionally, every meeting participant has the possibility to add the given meeting to their Google Calendar if only it has the final date set by the organizer.

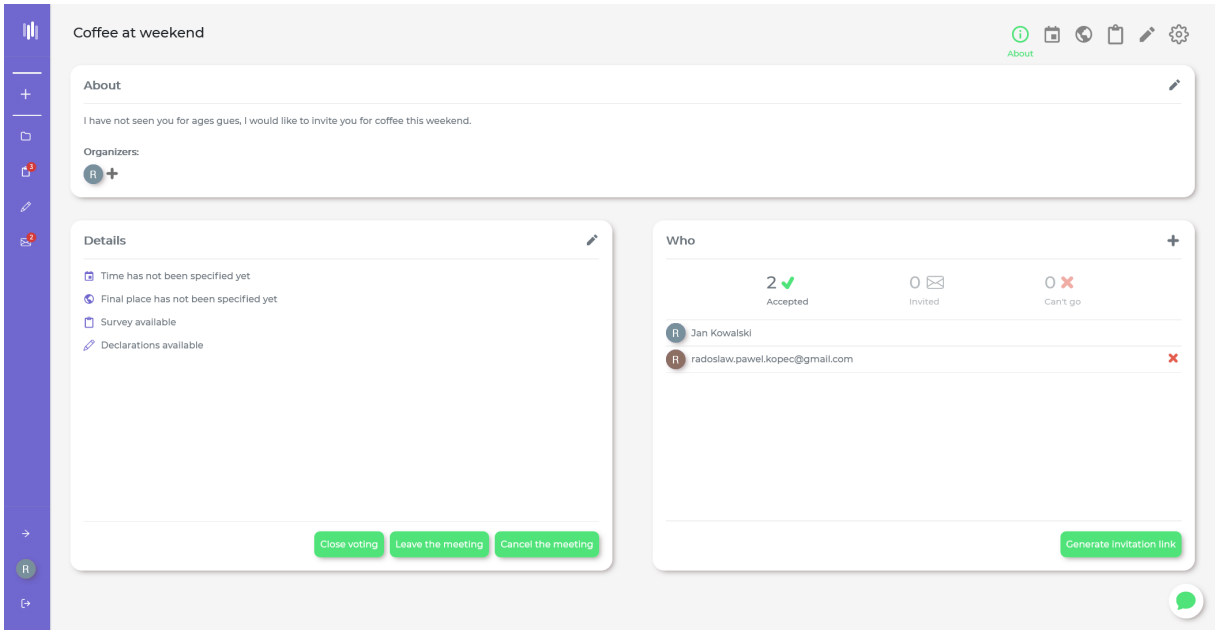

Figure 65: Meeting details view - section “About” - organizer view

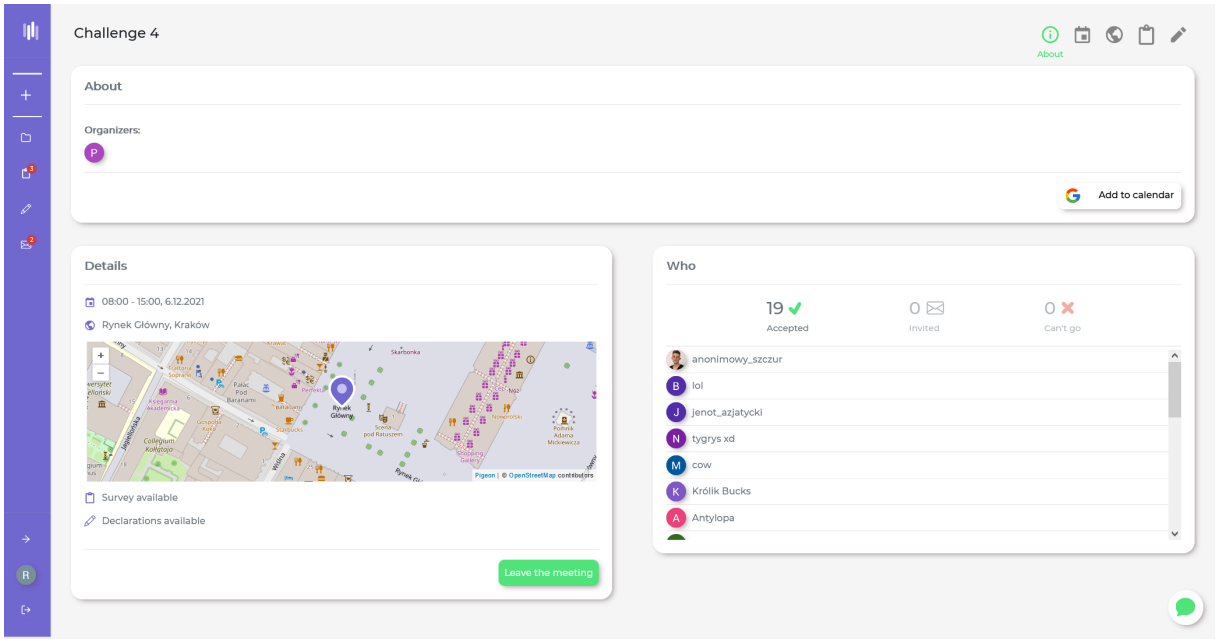

Figure 66: Meeting details view - section “About” - participant view

5.2.5.2. Meeting details view - section “Time”

The "Time" section is the second section responsible for time voting. The organizer may set the deadline for voting here, as well as a final date, taking into account all votes. The participants, on the other hand, can select their own time preferences here. User voices overlap in the calendar, creating darker areas where there are more of them and brighter areas where there are fewer. In this way, the organizer can choose a time that suits the preferences of the majority.

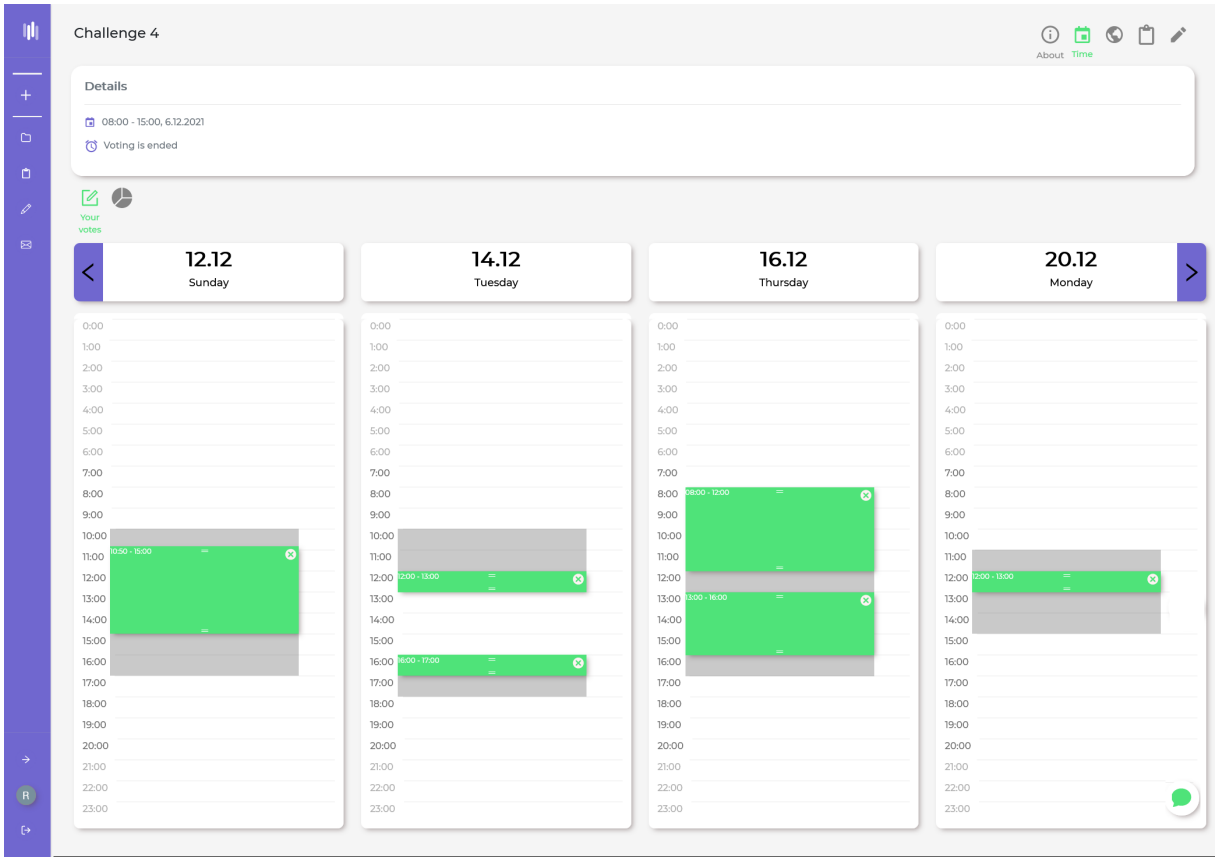

Figure 67: Meeting details view - section “Time” - current logged-in user votes

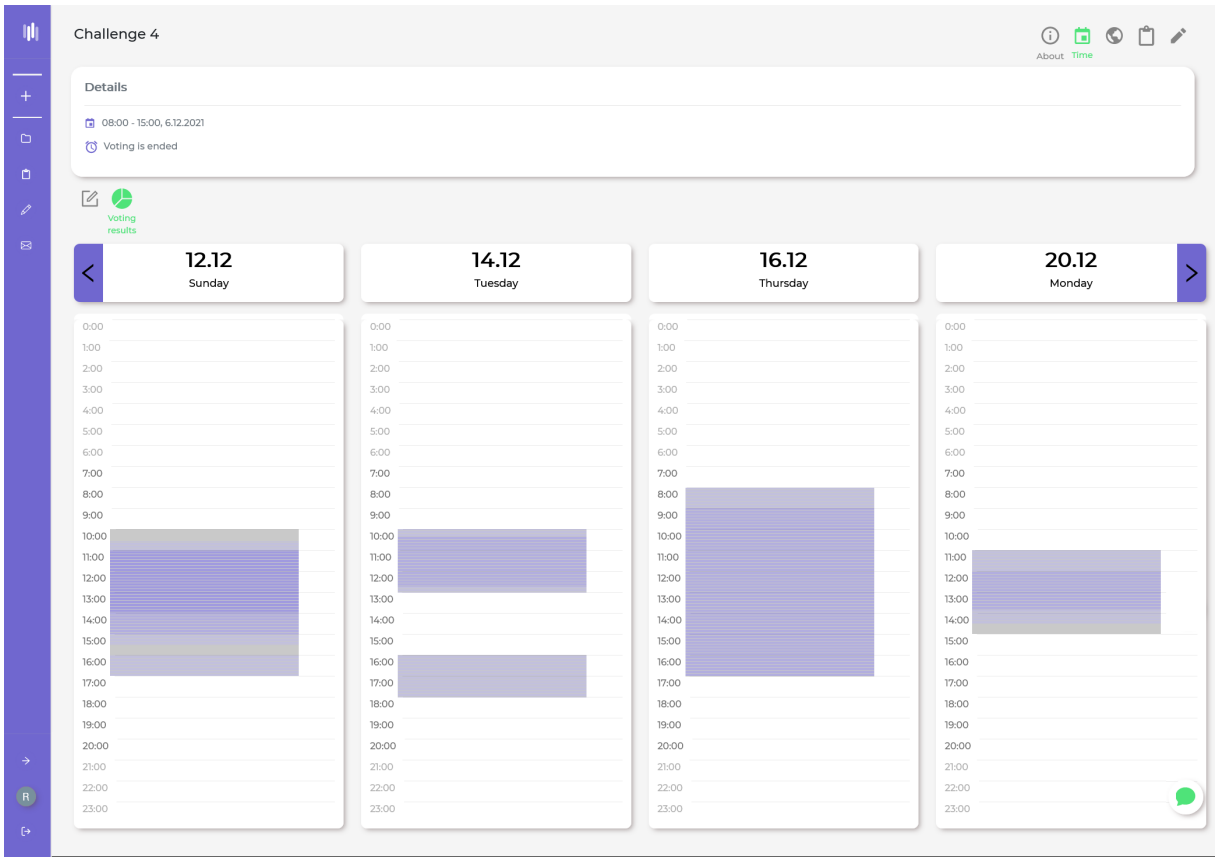

Figure 68: Meeting details view - section “Time” - all participants votes

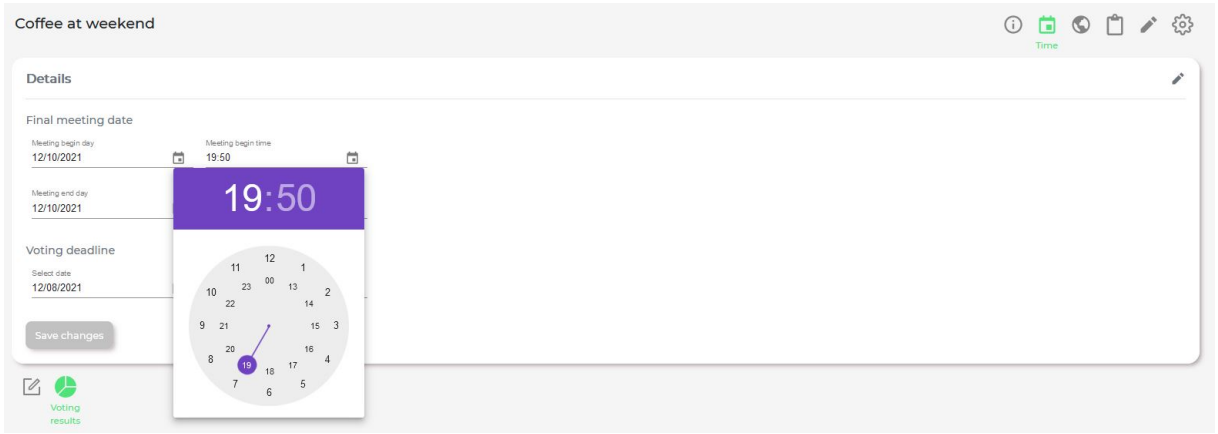

Figure 69: Meeting details view - section “Time” - final date and voting deadline edition

5.2.5.3. Meeting details view - section “Place”

The section “Place” is the third section which consists of some information related to the meeting place voting. All of the participants can vote for preferred places marked with pins on the interactive map. On the other hand, the organizer has the possibility to manage all the places by adding new or removing existing ones. Based on voting results the organizers can set the final place of the meeting. They can also allow other participants to view the results or to add new places to the poll.

At the bottom of the section, there is an additional table visible only for meeting organizers which allows them to manage available places.

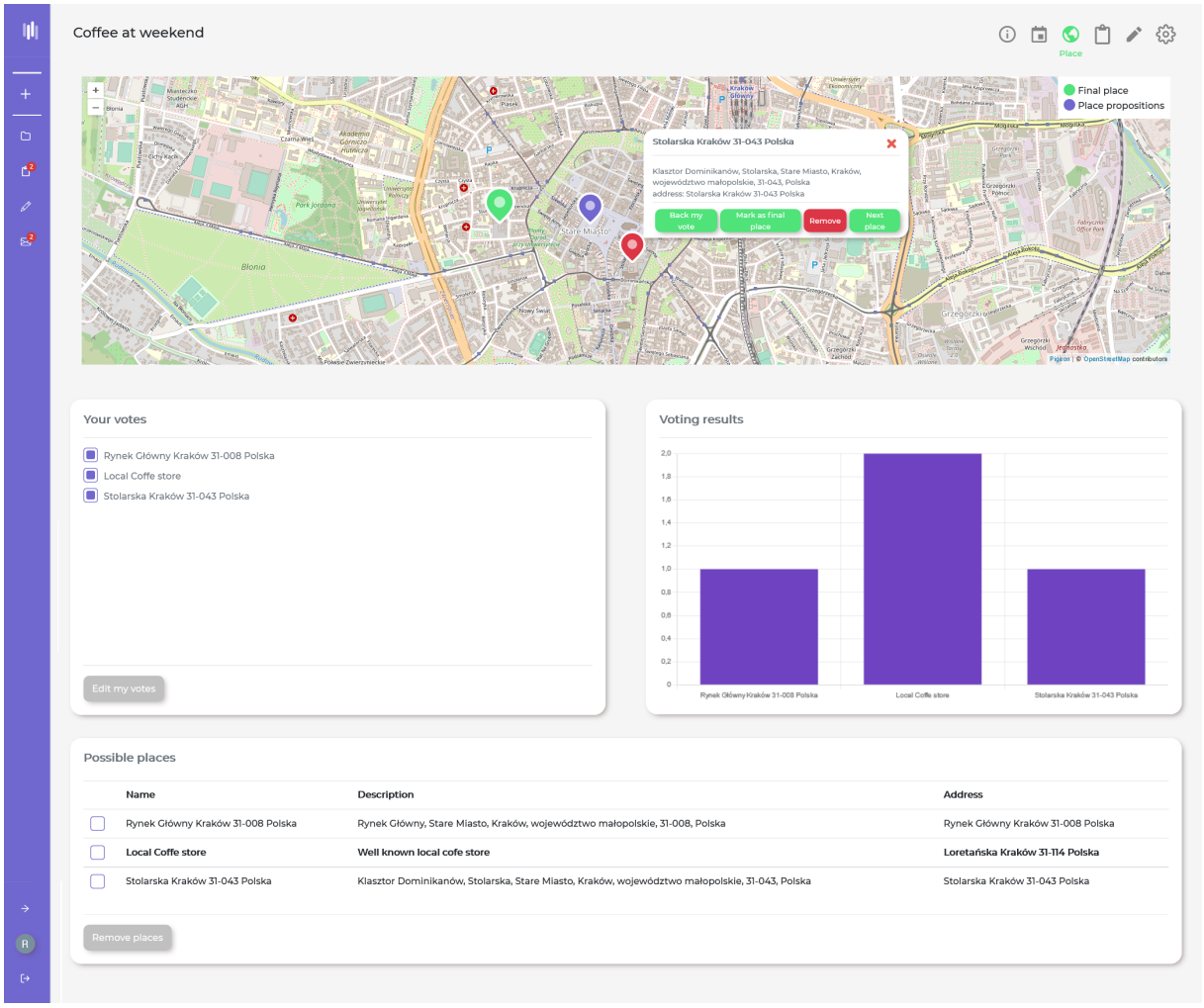

Figure 70: Meeting details view - section “Place” - organizer view

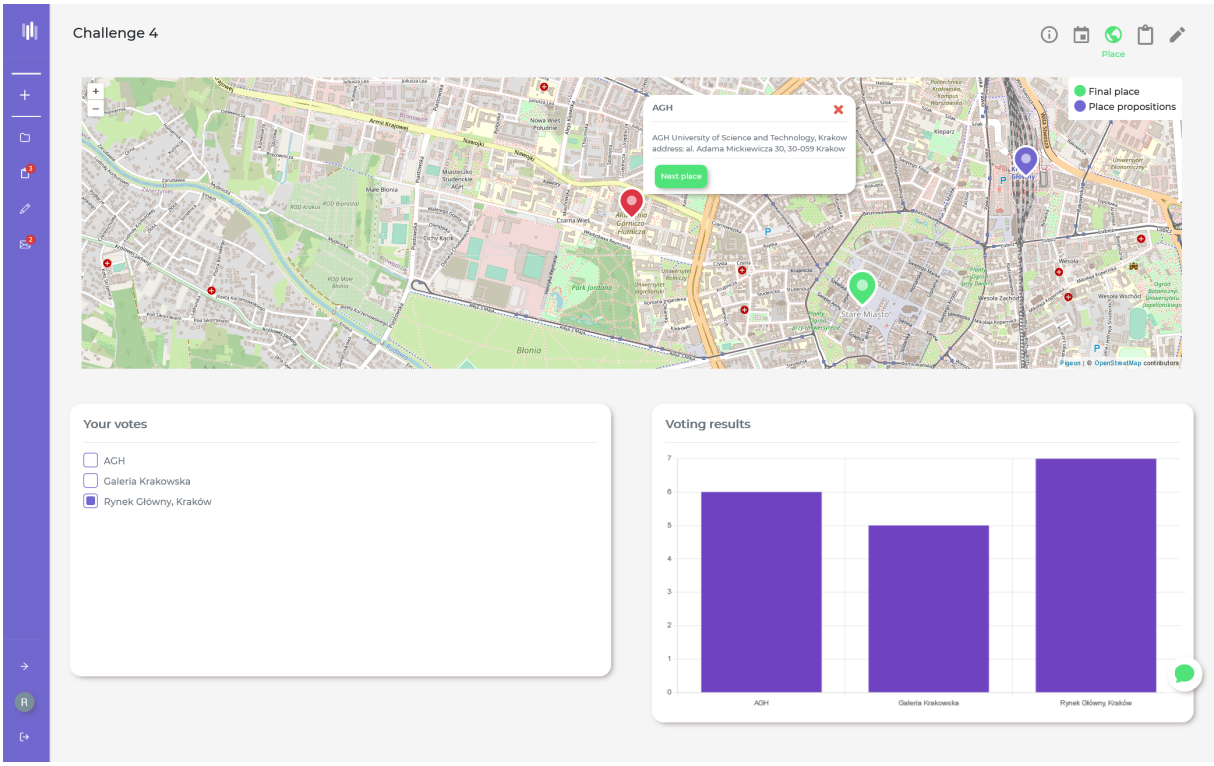

Figure 71: Meeting details view - section “Place” - participant view

The component visible for each place on the interactive map provides some most important actions depending on the role in the meeting. The organizer can use it to mark a place as a final one or to remove it from the poll. On the other hand, the meeting participant can use this component to vote for the place in a quick and convenient way. Additionally, all the pins on the map have different colors based on their state, so the users can easily distinguish the final place from other ones.

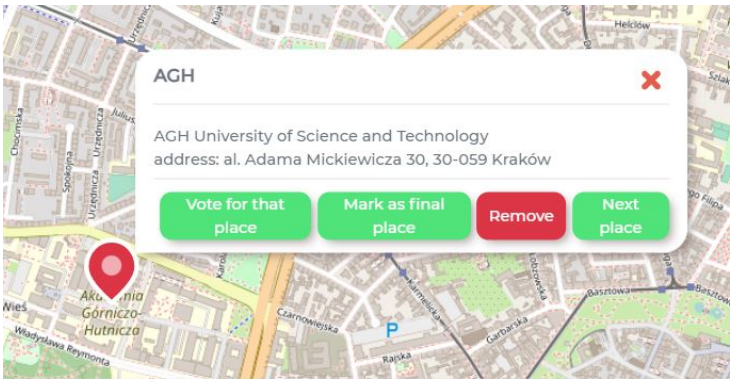

Figure 72: Meeting details view - section “Place” - place on the map from the organizer perspective

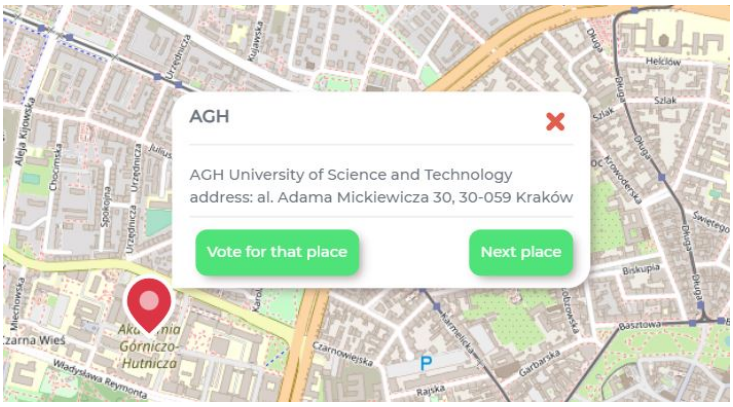

Figure 73: Meeting details view - section “Place” - place on the map from the participant perspective

5.2.5.4. Meeting details view - section “Survey”

The "Survey" section is the fourth section of the Meeting Details view responsible for managing surveys. In this section, participants can answer available questions or edit previously given answers, and if the organizer allows it, they can also view statistics. In turn, the organizer can edit the survey by changing its description, voting deadline, or adding and removing questions. The answer results vary depending on the type of question. You can see pie charts with the percentage of response data, the average value if the question was linear scale or individual responses.

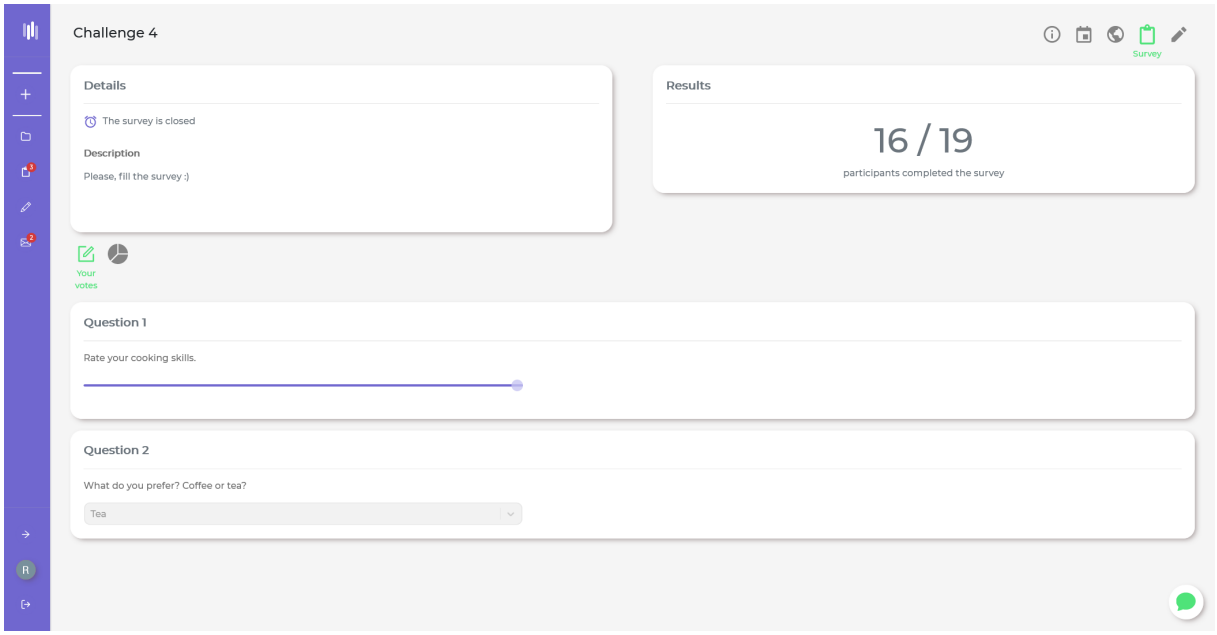

Figure 74: Meeting details view - section “Survey” - current logged-in user answers

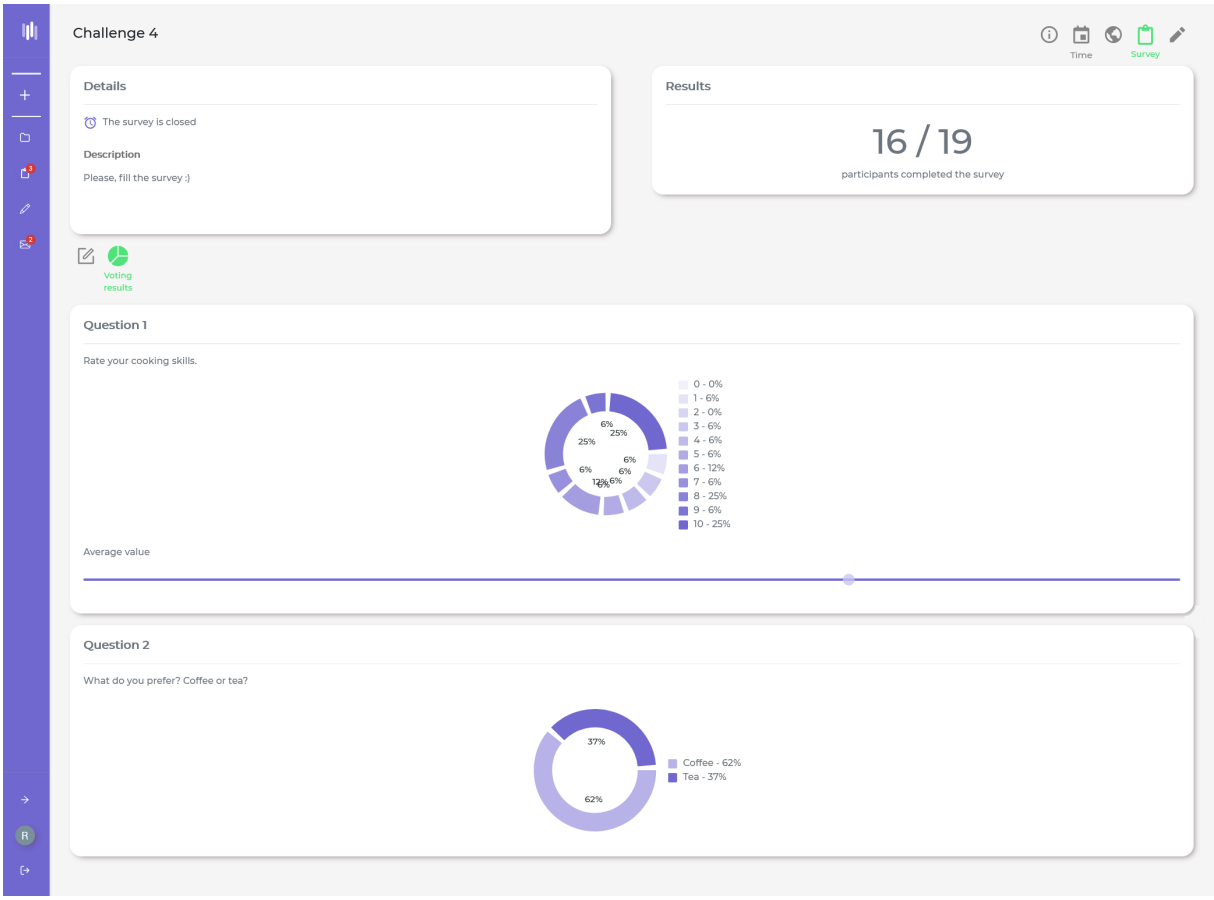

Figure 75: Meeting details view - section “Survey” - all participants answers with statistics

The organizer can also see who has already provided the answers. After clicking the "Show who filled the survey" button in the "Results" tab the modal with the list of users will be displayed.

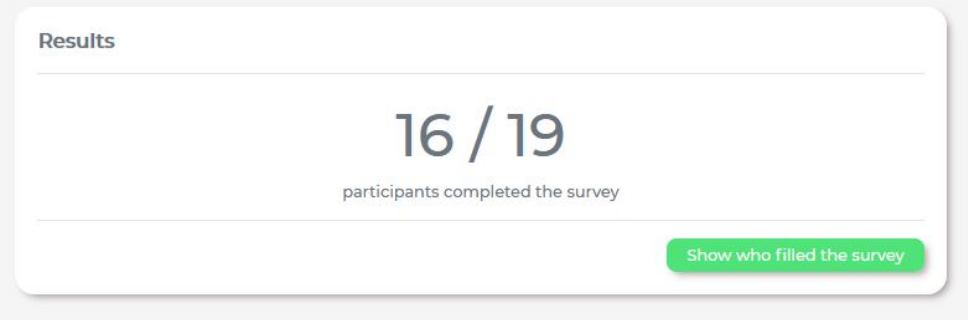

Figure 76: Meeting details view - section “Survey” - degree of completion of the survey

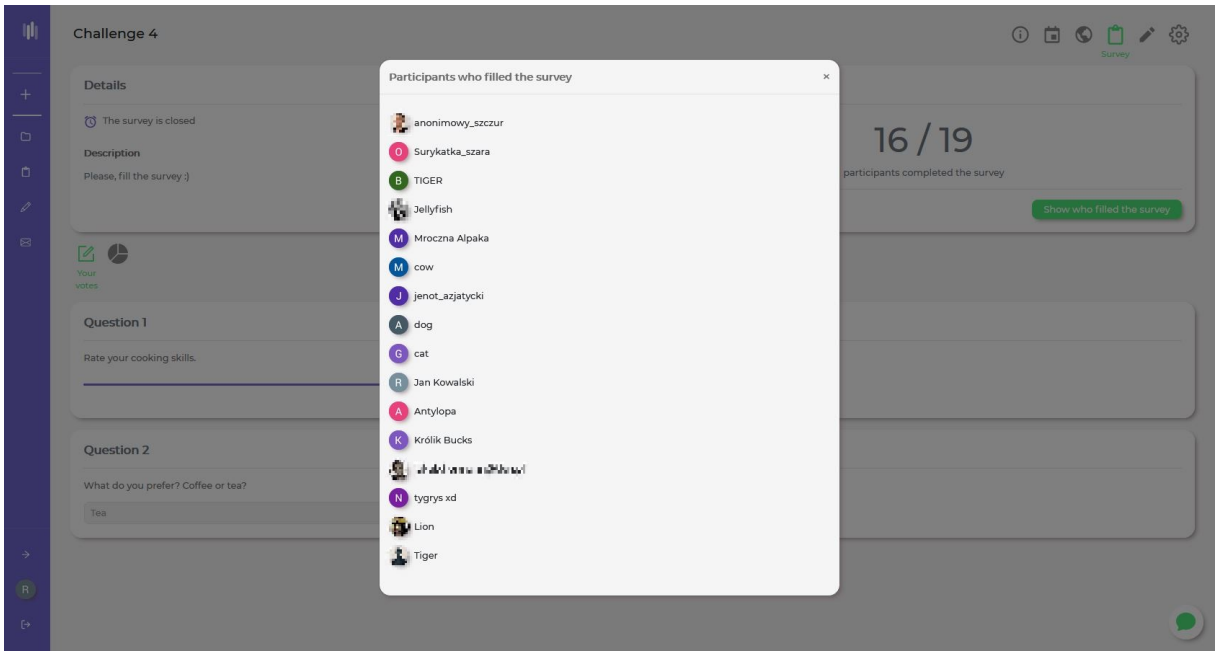

Figure 77: Meeting details view - section “Survey” - participants who filled the survey

5.2.5.5. Meeting details view - section “Declarations”

The “Declarations” section is a fifth section of the Meeting Details view responsible for displaying and creating user declarations. Participants can report their needs by creating a declaration card and assigning themselves to cards created by others. Users can only delete or modify their own declarations, while the organizer can modify/delete all of them.

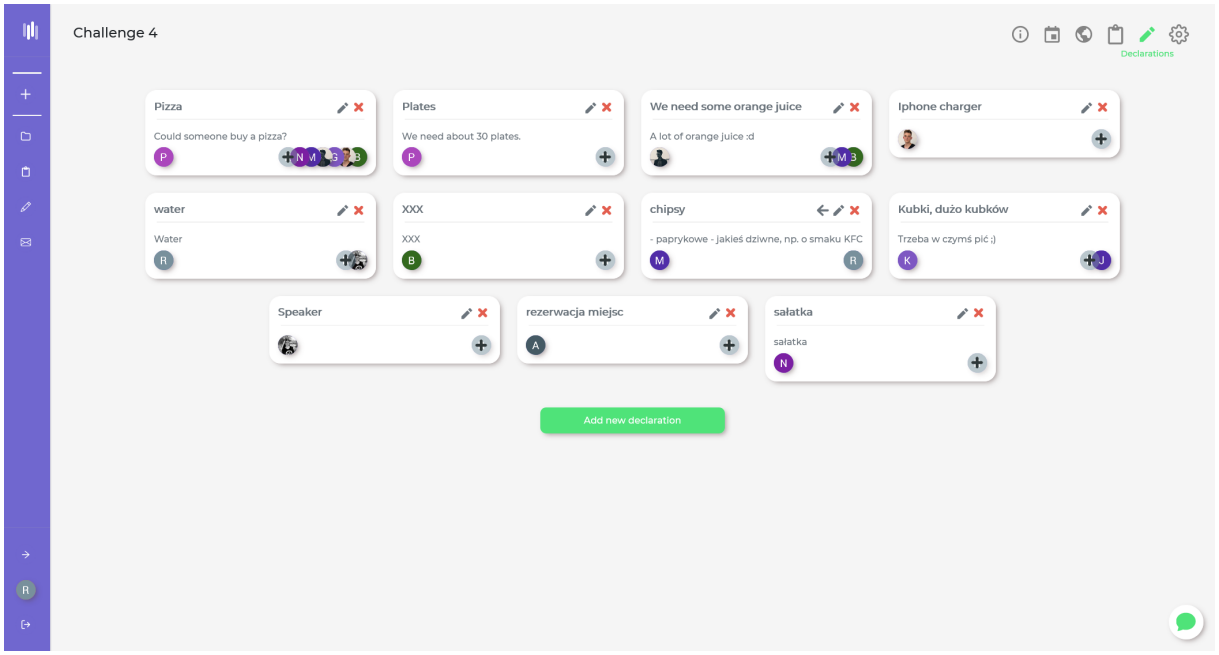

Figure 78: Meeting details view - section “Declarations” - organizer view

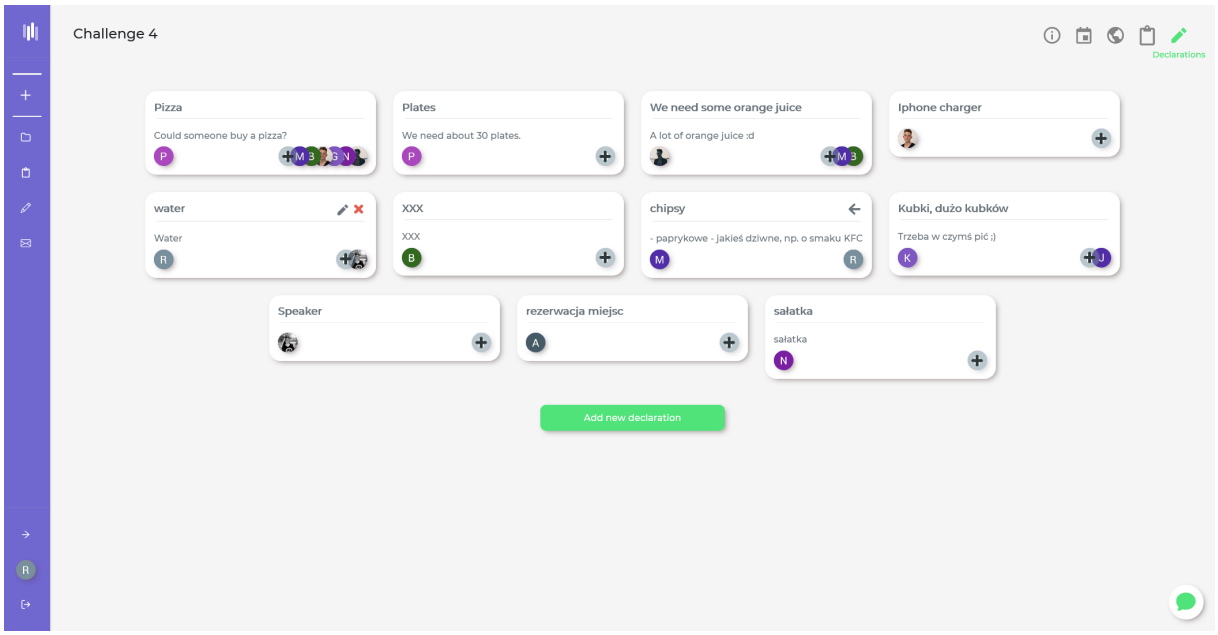

Figure 79: Meeting details view - section “Declarations” - participant view

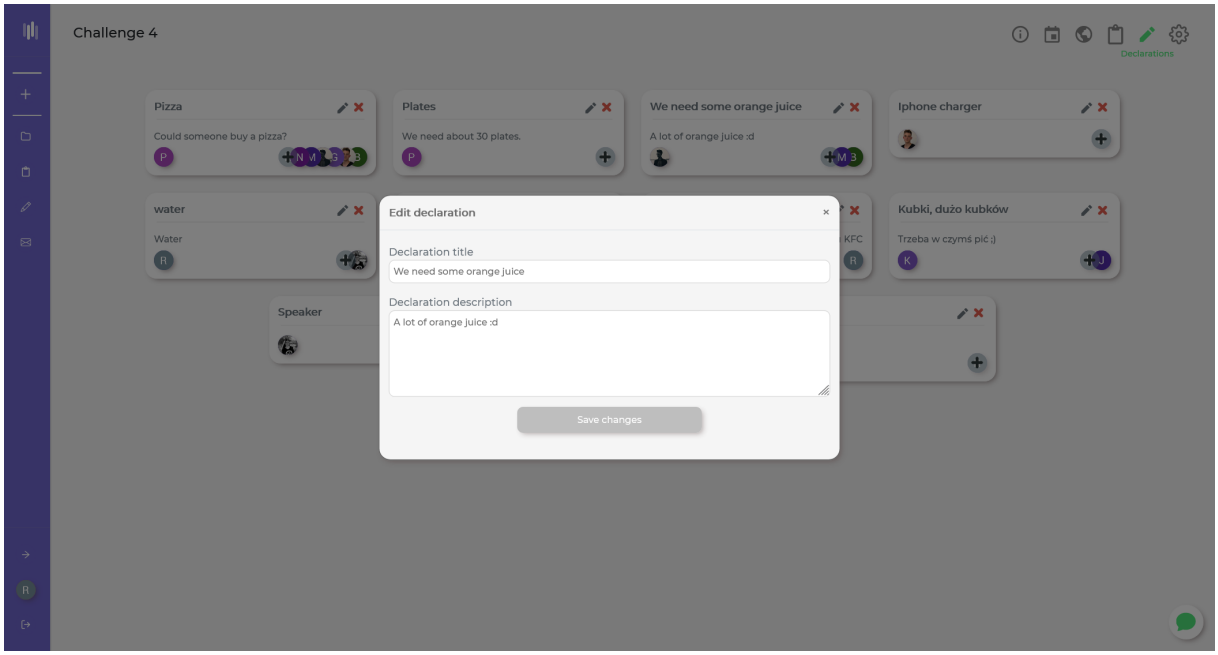

Figure 80: Meeting details view - section “Declarations” - declaration edition

5.2.5.6. Meeting details view - section “Settings”

The "Settings" section is the sixth section, visible only to the organizer, responsible for editing meeting settings. It is divided into several parts:

- general
- notifications
- places

General General settings allow organizers to specify whether participants can send meeting invitations and whether they should see the voting results for time, place, and polls. The meeting can also be deleted in this place.

Notifications In the notification settings, the organizer can send a notification to all meeting participants, set automatic reminders for completing a survey, for marking votes for the time of the meeting, and set notification before the meeting starts.

Places In the place settings, the organizer has the option of granting other participants the right to add new place suggestions.

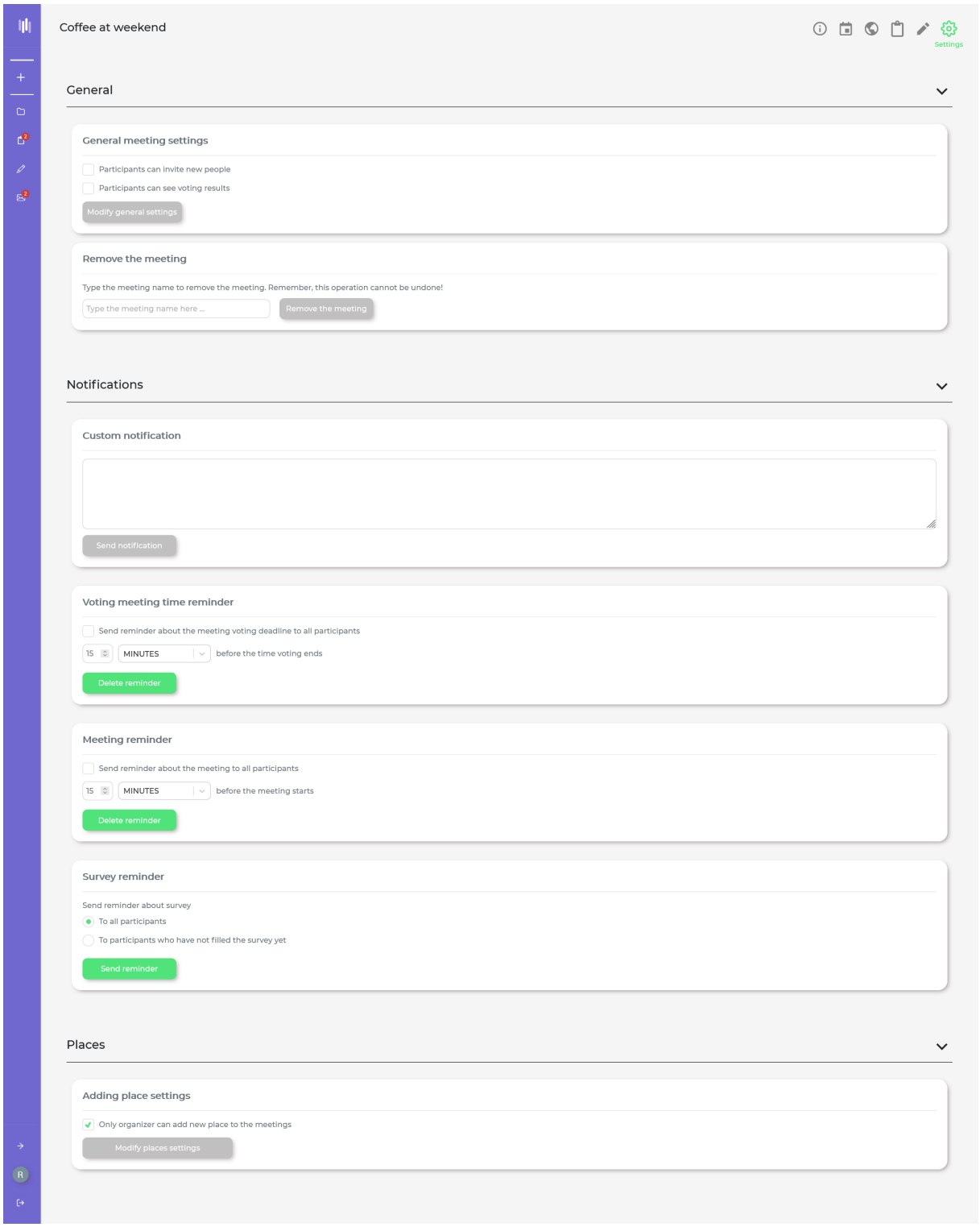

Figure 81: Meeting details view - section “Settings”

5.2.5.7. Meeting details view - chat

Finally, it is worth mentioning the chat, which allows meeting participants to quickly exchange messages with each other. The purpose of our application was to gather all the most important functionalities of scheduling a meeting in one place, so there was also a chat, which was created with the participants' convenience in mind.

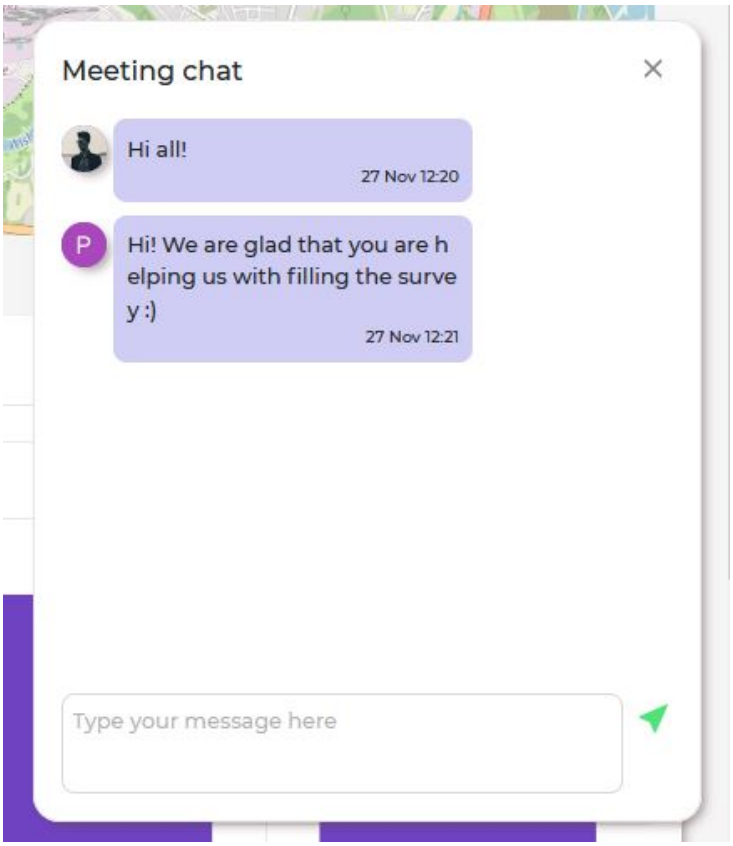

Figure 82: Meeting details view - chat

5.2.6. All meetings view

The view that can be accessed via the "Meetings" link in the navigation bar is the view containing all meetings, i.e. those organized by the currently logged-in user, those in which he participated and which have ended, as well as those that will take place. They are presented in the form of tables containing information such as the name of the meeting, the list of organizers, the date, and the current status. In the table with past meetings, there is an additional option to delete them and in each of the tables, it is possible to easily search for a meeting by keywords. Another important function is that clicking on a table row will redirect you to the given meeting.

The business justification for this view is that the user would certainly like to have all the meetings gathered in one place, as this allows them to be accessed faster and easier to find. However, it was a challenge to display the tables correctly in the mobile version, as they did not fit on the narrow screen. In order to solve this difficulty, we decided to scroll them vertically. The desktop version of this view is presented below.

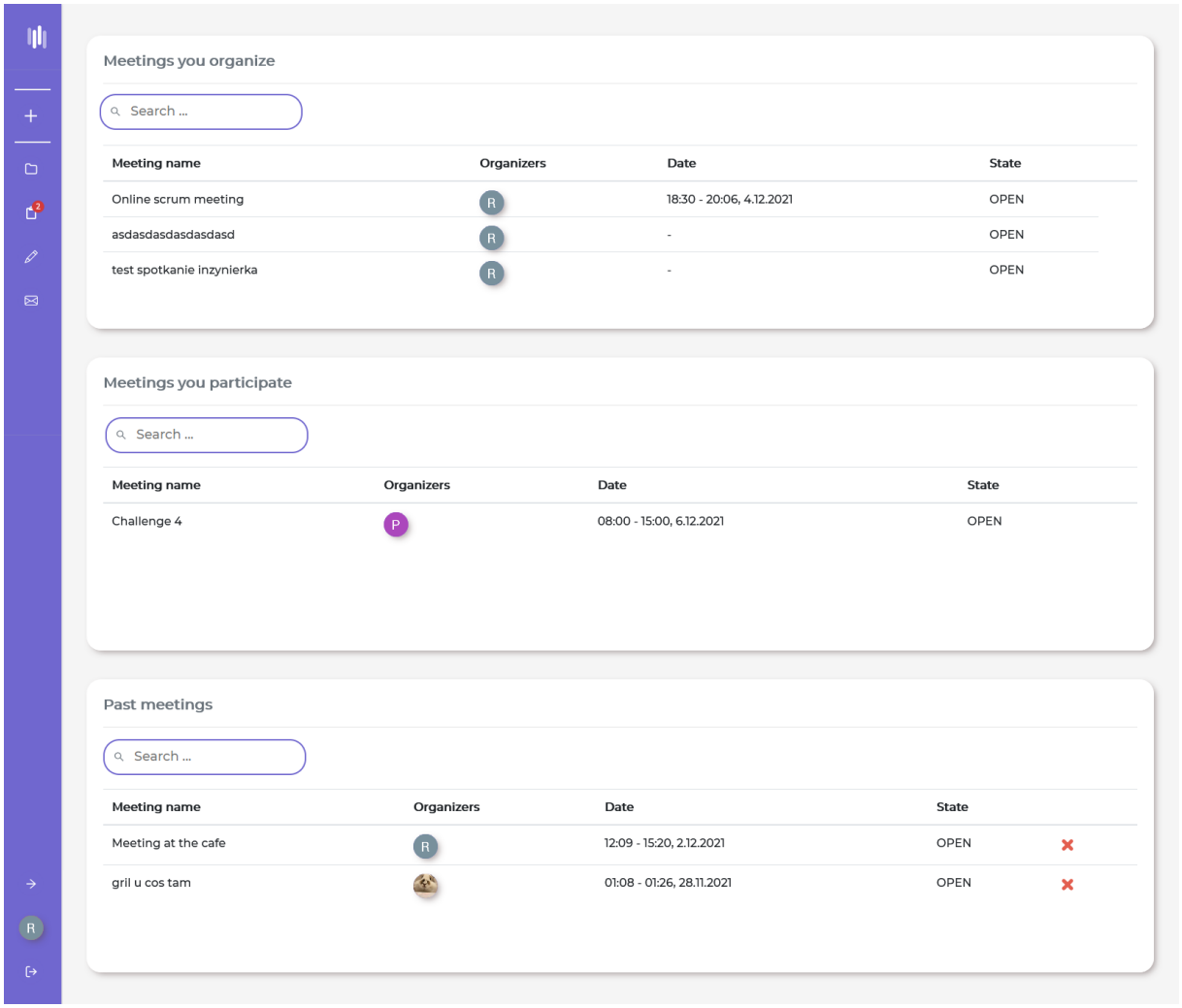

Figure 83: All meetings view for desktop devices

5.2.7. Unfilled surveys view

The view that can be accessed via the "Surveys" link in the navigation bar is the view that contains the surveys that have not yet been completed. The most important information presented here is the meeting name, its organizers, the deadline for completing the survey, and the number of available questions. When the user has unfilled questionnaires, a small red circle appears in the navigation bar with the number of them. This is a psychological procedure as it encourages people to click on a tab and take action on the survey. A similar technique is used by social networks such as Facebook or Instagram. The aforementioned view in the desktop version is presented below.

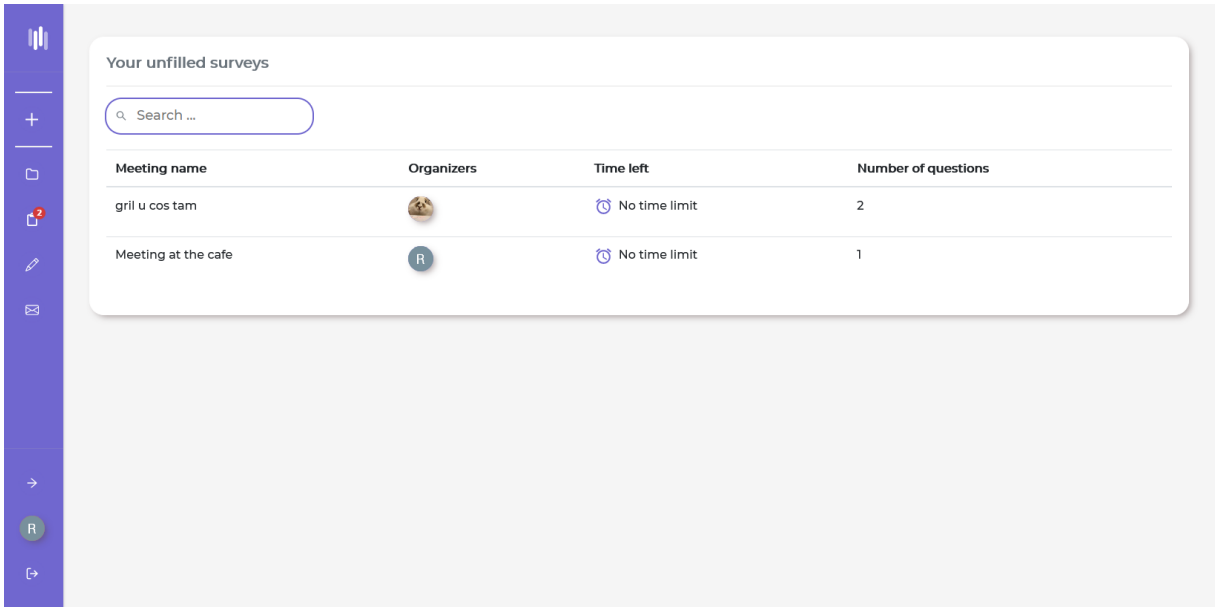

Figure 84: Unfilled surveys view for desktop devices

5.2.8. All declarations view

The declarations view contains a table with both the declarations created by the logged-in user and those to which he is assigned. As a result, he can immediately see people related to his declarations. Clicking on a line is redirecting to a meeting from which the given declaration comes. The information contained in the table is the name of the meeting, the title of the declaration, the person who created the declaration, and who is assigned to it. The aforementioned view in the desktop version is presented below.

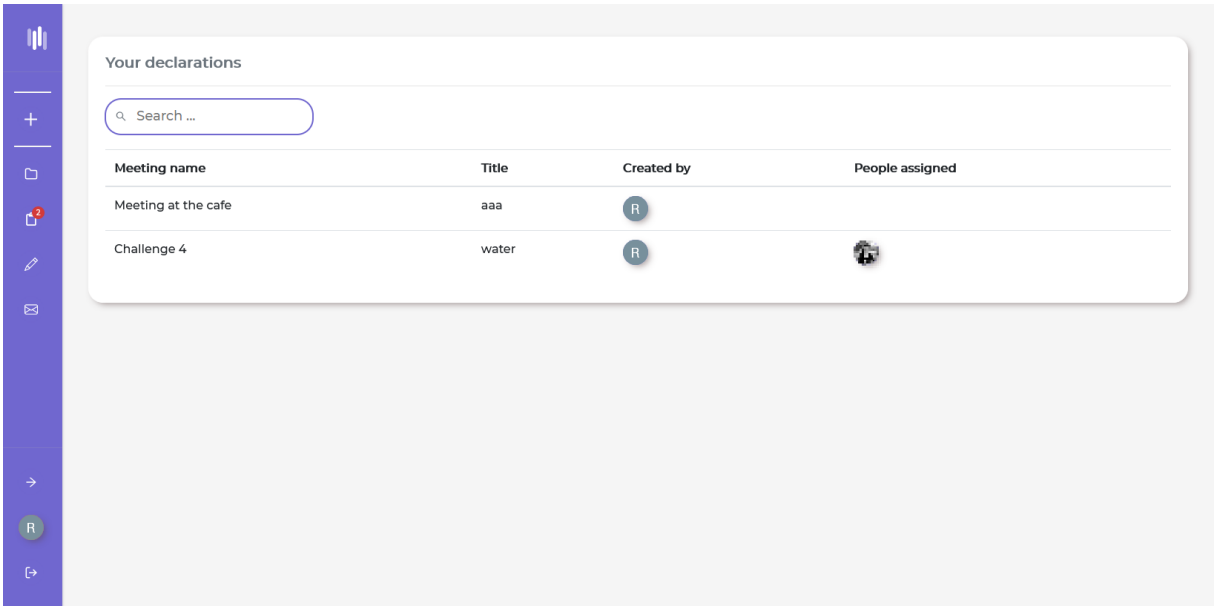

Figure 85: All declarations view for desktop devices

5.2.9. Invitations view

The invitations view shows a table with the active invitations of the currently logged-in user, which contains basic information such as the name of the meeting and the person who sent the invitation. The last column contains buttons for accepting or rejecting them. Moreover, when a user receives an invitation, a red circle with the number of active invitations appears in the navigation above the envelope icon, prompting the user to take action. The aforementioned view in the desktop version is presented below.

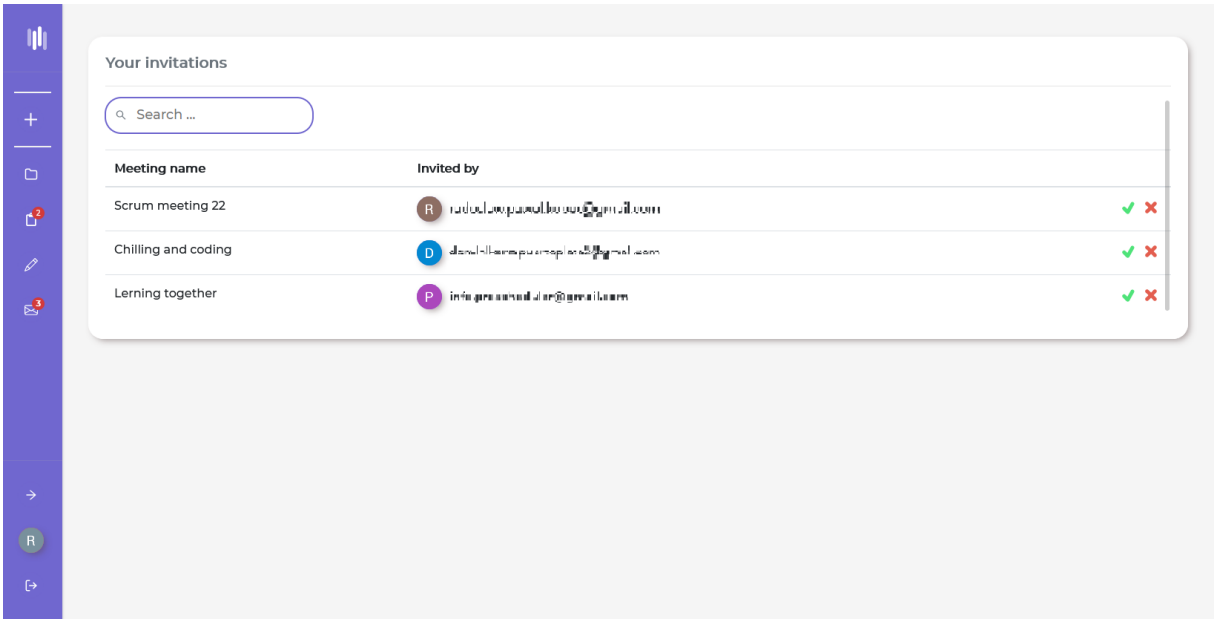

Figure 86: The invitations view for desktop devices

5.2.10. Meeting link invitation view

The last subpage is the meeting link invitation view, which is displayed after entering the invitation link. It provides basic information about the meeting, such as its name, when it will take place, and from whom the invitation was received. After clicking the “Join” button, the user joins the meeting and is redirected to its details view. However, if the user is not logged in, he will be redirected to the login page. The aforementioned view is presented below.

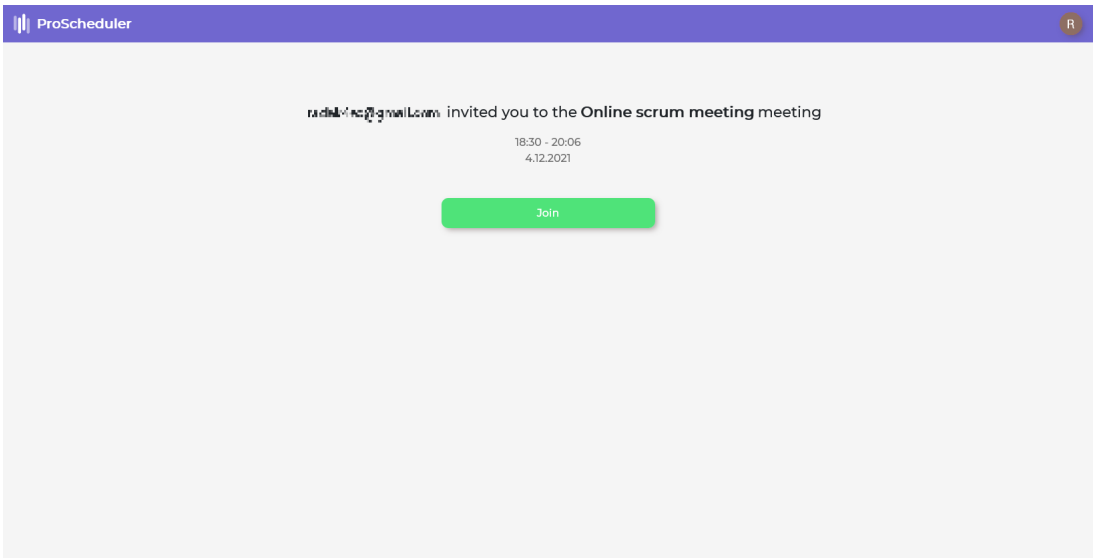

Figure 87: Meeting link invitation view

5.3. Summary

After we conducted the survey and showed the product to the final group of users, we received positive feedback. The application raised huge interest among people testing it and the provided system fulfills all the requirements specified by the client with a surplus. We delivered all required functionalities and added many additional, that enhance the product. We are really glad about the result we achieved and the decisions we made, which were accurate and resulted in a well-functioning application that can be successively improved in the future. We are particularly satisfied with the good-looking and intuitive UI we delivered and the rapid and optimized back-end side. Additionally, we learned a lot during the development process, including:

- creating a microservice architecture
- managing a virtual machine
- hosting application
- frameworks - React, Spring-Boot
- working in Scrum
- teamwork

It was a really fantastic journey to create the whole system from scratch and we are more than satisfied with the final outcome.

5.4. Further plans

The resulting application is currently fully functional in terms of creating and managing meetings - there are provided essential features like time or place voting, surveys, or declaration support. However, during product development, we have come to some more ideas on how the application can be improved and what next features can be provided. They are related both to technical optimizations (e.g. to bring end-users faster and more reliable solutions) and additional functional features which can be a brilliant extension to the current ones to provide an even better user experience.

5.4.1. Functional improvements

5.4.1.1. Creating a meeting based on the previous one

Creating a meeting based on the previously existing one is the additional feature that has been proposed as a “nice to have” one by the client during the late product development. Unfortunately, we have not succeeded in implementing it due to other higher priorities. This feature would allow users to create a new meeting based on the configuration of the other existing one. It would significantly reduce the time that the user has to spend on configuring the new meeting (e.g. creating a survey with multiple questions from scratch) - which is certainly a huge benefit regarding the user experience aspect.

5.4.1.2. Further integration with Google Calendar

One of the features our application provides is the integration with Google Calendar. Currently, there is only one-way integration that allows users to add the given meeting to it. In the future development, we would also want to provide the possibility to show all user Google Calendar events in the user dashboard because the key objective of our application is to provide an integrated environment to reduce the necessity of usage of other applications to the minimum. Certainly, it would also be a significant user experience benefit as the user would not need to manually check their time availability through multiple applications.

5.4.1.3. Tutorials and tooltips for new users

We have done our best to provide as intuitive user interface as possible in our application. One of our next steps would be to provide custom interactive user guides for each of the features the application offers. The example tutorial would navigate the user step by step through creating a new meeting wizard or through a meeting details view to show all the key features they can use to manage the meeting in the most efficient way. Such a solution would be really helpful especially for new users who use the product for the first time and they are not so familiar with all its aspects and features.

5.4.1.4. Cost splitting module

The application can be easily extended by additional sets of features which can be called modules. It is achievable thanks to the designed system architecture based on microservices.

Therefore, we have also plans to provide an additional module that would allow users to share some meeting-related expenses with other people. There are multiple applications currently on the market (like Splitwise or Tricount), however, we want to provide users an integrated solution to have all possibly useful functionalities in one place - which is again a key advantage of our product.

5.4.1.5. User dashboard highly integrated with notifications

Our product has also implemented a well-developed notification system - mainly using email messaging. Therefore, one of the next steps in eventual further development would be to provide enhancements to the user dashboard to make it more dynamic and highly connected with different types of notifications - e.g. the ones related to invitations, created declarations, or time/place voting deadlines. Such integration would allow users to easily manage multiple meetings because all essential information would be located at one place without the necessity to navigate over each meeting separately.

5.4.1.6. Dedicated mobile application

One of the next steps of our product development would be designing and creating a dedicated application for mobile devices which would be an alternative for the current web one - especially for users using smartphones or tablets. The dedicated mobile app would make the designed notification system be used in a more efficient way because there would be the possibility to implement custom notifications with native solutions provided for mobile devices. Furthermore, multiple parts of the applications could be redesigned to follow solutions specific to mobile apps like using native components or navigation systems. Following such an approach would definitely make our product more popular - especially due to the fact that most of the users who participated in UX tests used mobile devices to accomplish the tasks.

5.4.2. Technical improvements

5.4.2.1. Moving application to the cloud

Currently, the application is running on our own server bought to the need of developing the system on ovhcloud.com. This server is only a temporary solution because with the higher load it will not guarantee such reliability as a cloud. Cloud providers offer a number of tools that ensure reliable performance and scaling so that the application will profit with fault tolerance and when a machine fails then the system will still be functional. There are load balancers available that will ensure high–availability due to the replication.

Cloud providers offer environments that work under the control of Kubernetes, such as Amazon EKS. This solution will allow us to use the containerization benefits and manage our application with the use of a container-orchestration engine – Kubernetes. Moving the application to the cloud is definitely one of the next crucial steps that need to be made to ensure reliable working in the production environment.

5.4.2.2. Monitoring

Monitoring is a significant element of a system that allows to localize the potential bottlenecks and observe the behavior of the application. Metrics help with identifying the places that need improvement and finding all interesting details of the working application as the number of requests per second. It is very important because with the use of them we can recognize when the application starts working in an unpredictable and unwanted way.

5.4.2.3. Distributed tracing and log gathering

Our application operates in a distributed environment, therefore a very important element is distributing tracing - correlation of the operations executed in particular services. Due to this solution, it will be possible to logically connect operations executed in different microservices with each other and identify potential failures. We need logs from each service to achieve it and it can be done with the use of ELK stack [14] - Elasticsearch, Logstash, Kibana. Logstash sends logs from connected logfiles to Elasticsearch, then with the help of Kibana, we can browse through them to identify failures. Logs are a significant part of every application, when a failure occurs they are the first thing developers look at, so its gathering is an important part that needs to be introduced in the future.

5.4.2.4. Specifying own database for each service

Currently, there is only one database for all services in our system. The root cause of this decision was that our server does not have a sufficient amount of resources to keep several instances running. As a target we want every service to have its own database to rump the efficiency of application up. Additionally, we could consider selecting different database types and different services depending on the needs and model that the service operates on.

5.4.2.5. Integrating with Sendgrid

As of this moment, we have our own solution implemented to send the emails to users, but we are aware that this is not a permanent way of doing it as it can not guarantee to send many concurrent emails to huge amounts of receivers. Sendgrid [52] is a specific service that offers the functionality of sending emails on-demand, providing perfect performance. Unfortunately, this is a paid solution and therefore we have not introduced it yet in our project.

References

[1] Activemq. `https://activemq.apache.org/`.

[2] Adobexd. `https://www.adobe.com/products/xd.html`.

[3] Amazon eks. `https://aws.amazon.com/eks/`.

[4] Angular. `https://angular.io/`.

[5] Artifactory. https://jfrog.com/artifactory/.

[6] Bitbucket. https://bitbucket.org/product.

[7] Chart.js. https://www.chartjs.org/.

[8] Circleci. https://circleci.com/.

[9] Date picker from material-ui. https://material-ui-pickers.dev/demo/datepicker.

[10] Discord. https://discord.com/.

[11] Docker. https://www.docker.com/.

[12] Dockerhub. https://hub.docker.com/.

[13] Doodle. https://doodle.com/en/.

[14] Elk stack. https://www.elastic.co/elastic-stack/.

[15] Facebook. https://www.facebook.com/.

[16] Git. https://git-scm.com/.

[17] Github. https://github.com/.

[18] Github actions. https://github.com/features/actions.

[19] Gitlab. https://gitlab.com/.

[20] Google accounts. https://www.google.com/account/about/.

[21] Google calendar. https://www.google.com/calendar/.

[22] Google forms. https://www.google.pl/intl/pl/forms/about/.

[23] Gradle. https://gradle.org/.

[24] Hibernate. https://hibernate.org/.

[25] Java. https://www.java.com/.

[26] Javascript. https://www.javascript.com/.

[27] Jenkins. https://www.jenkins.io/.

[28] Jira. https://www.atlassian.com/software/jira.

[29] Junit5. https://junit.org/junit5.

[30] Kafka. https://kafka.apache.org/.

[31] Kotlin. https://kotlinlang.org/.

[32] Kubernetes. https://kubernetes.io/.

[33] Lombok. https://projectlombok.org/.

[34] Material-ui. https://mui.com/.

[35] Maven. https://maven.apache.org/.

[36] Micronaut. https://micronaut.io/.

[37] Microsoft teams. https://www.microsoft.com/en/microsoft-teams/.

[38] Mockito. https://site.mockito.org/.

[39] Mongodb. https://www.mongodb.com/.

[40] Ms sql. https://www.microsoft.com/pl-pl/sql-server.

[41] Mysql. https://www.mysql.com/.

[42] Neo4j. https://neo4j.com/.

[43] Nginx. https://www.nginx.com/.

[44] Openstreetmap. https://www.openstreetmap.org/.

[45] Postgresql. https://www.postgresql.org/.

[46] Python. https://www.python.org/.

[47] Rabbitmq. https://www.rabbitmq.com/.

[48] React. https://reactjs.org/.

[49] React-bootstrap. https://react-bootstrap.github.io/.

[50] Redux. https://redux.js.org/.

[51] Scala. https://www.scala-lang.org/.

[52] Sendgrid. https://sendgrid.com/.

[53] Spring. https://spring.io/.

[54] Spring boot. https://spring.io/projects/spring-boot.

[55] Strawpoll. https://strawpoll.com/en/.

[56] Travisci. https://travis-ci.org/.

[57] Typescript. https://www.typescriptlang.org/.

[58] Vue.js. https://vuejs.org/.

[59] Whenavailable. https://whenavailable.com/.