Evented

TECHNISCHE UNIVERSITÄT DARMSTADT

A media based Event application to meet new people and have fun

Telecooperation Lab

Simon Jungherz
Laurenz Kammeyer
Jonas Milkovits
Jannik Schmidt
Frederick Wichert

Winter Term 2022/2023

Internet - Praktikum Telekooperation

Contents

1 Motivation

Going into this project we were all united to create something that we would all like to use ourselves. As everyone of us likes to meet new people and the current offers for applications with local communities are really limited, the choice for Evented was an easy one. We had some other ideas but decided that supplementing local meetups with media would be the best one.

An application that actually forces the people to leave their own house to join other people is also something we strongly support, especially following the current/past pandemic. Everybody who wants to meet new people, party and have some fun should have the option to do just that. That's what we are trying to achieve with Evented!

The app itself is focused around media files, like images, videos and livestreams. The main way to discover new events is by taking a look at those media files that people took at those events. This helps the user to get a feeling for the event and helps them to decide if they want to join.

2 Technical Overview

2.1 Technologies & Services

We used a lot of different technologies and services to develop our application. In the following you can find an overview with a short description of each technology/service.

- General
  - Node.js + npm: Node.js is a well-known (backend) JavaScript runtime. It is used in combination with npm, a command line tool to manage dependencies for your project. We used it in our front- and backend.
  - TypeScript: TypeScript is a typed programming language built on top of JavaScript. It transpiles to JavaScript and provides static typing to it. This helped tremendously to spot bugs in our application. As we all had no real experience with it, it was a bit rough in the beginning.
  - Prettier: As we all know everybody likes to write their code in a different way. Integrating Prettier in our precommit pipeline allowed us to keep our codebase clean by automatically formatting our code when creating git commits.
  - ESLint: A well-known static code analysis tool for JS/TS applications is ESLint. We used it to spot some bugs and errors earlier and to keep our code as bug free as possible. We also integrated it into our precommit pipeline.
- Frontend
  - Figma: Figma is a great tool to create quick and good looking mockups for your ideas. It is considered high fidelity and we mostly used it at the beginning of the project to align our design ideas.
  - React Native: React Native is an adaption of React for mobile applications. It uses native platform functions where possible and is one of the most popular frameworks for cross-platform applications. We decided to use React Native, because we wanted to make our application cross-platform and already had some experience with React.

  - Expo: Expo is built on top of React Native, and helped us to make the whole development process a lot easier. Using it together with the android app 'Expo Go' allowed for fast and streamlined development. It provides lots of useful libraries and works well with React Native.
  - react-native-nodemediaclient: A module to create and display RTMP video streams in React Native.

- Backend
  - Docker/docker-compose: Docker is used to separate different services into containers, which we used with docker-compose on top of that for easier orchestration. Docker provides isolation between the containers and the host, and makes configuration/migration easier.
  - Caddy: Instead of a classic choice like nginx, we went with Caddy as our reverse proxy, both to try some new technology and since configuration as well as certificate management is quite intuitive.
  - Express: Express is a really lightweight backend framework to create APIs in Node.js. As we all really like the framework we decided to also use it for this project.
  - PostgreSQL: PostgreSQL is one of the most commonly used database management systems. For local development we mostly used SQLite but our live server uses PostgreSQL.
  - Sequelize: As we all already knew how to work with SQL, we decided to try something new and integrated the object relational mapper (ORM) Sequelize to our project. It provides another layer of abstraction to the communication between our web and database server.
  - passport.js: We used this library for authentication purposes. It's a lightweight library that integrates really well in Express and offers a lot of different strategies to implement authentication.
  - ffmpeg: ffmpeg is a popular collection of libraries to handle media files. We use it to process videos in our backend to create different qualities.
  - sharp: sharp is a Node.js module for simple and fast image processing. We use it to compress and resize user uploaded images.
  - Jest: We used Jest for automated testing, ensuring that all API endpoints work as intended. This is also set up to automatically run on every commit in GitLab CI.
  - node-media-server: A module for handling live media streams in Node.js. We use it to accept and broadcast live video streams.

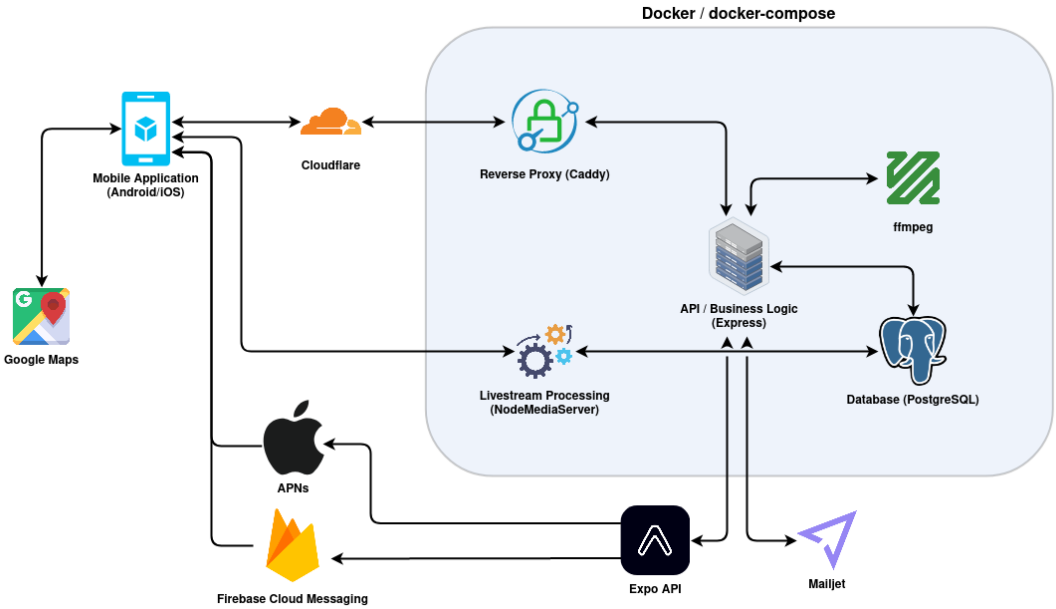

Figure 1: Architecture

2.2 Backend

2.2.1 Database

As already seen in the technologies section, we are using PostgreSQL as our database. We are however not writing raw SQL to communicate with the database, but instead decided to add another layer of abstraction and use an object relational mapper. Here we are using the popular ORM Sequelize. We decided to use Sequelize as we are all rather familiar with SQL and wanted to learn something new.

The following ER-diagram represents an overview of all our tables. Here you can clearly see how heavily intertwined events and users are.

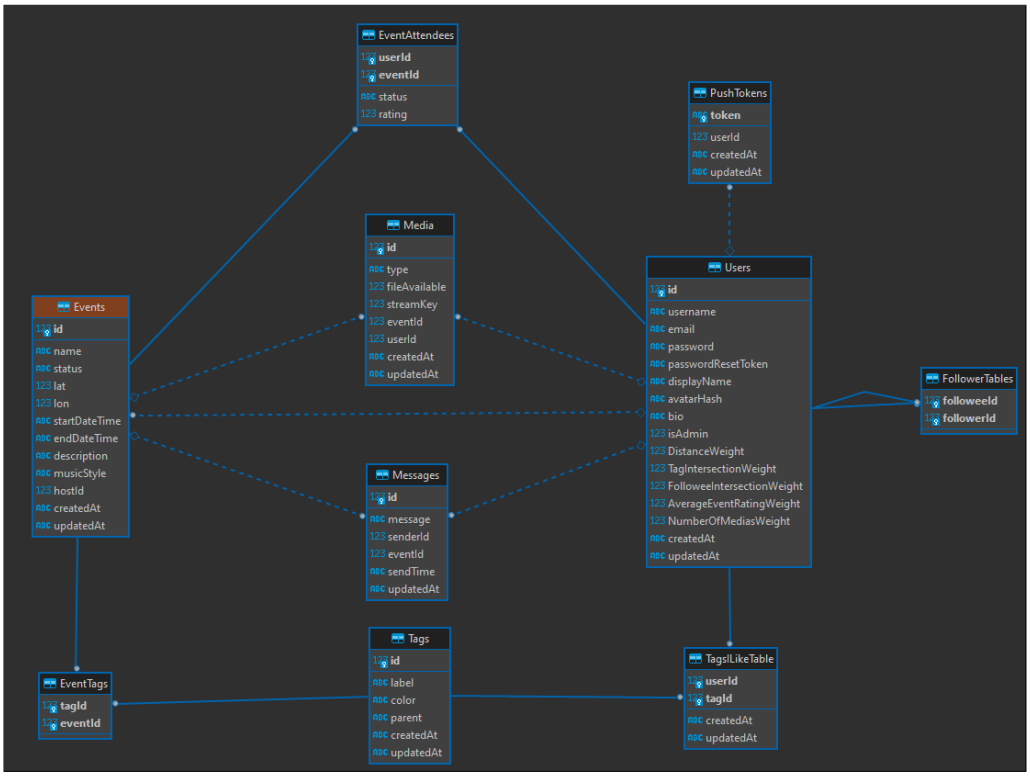

Figure 2: ER Diagram

2.2.2 Authentication

As previously mentioned we are using the library passport.js in combination with bcrypt to create a secure authentication service/storage. Whenever the user creates an account or changes their password, we are using bcrypt to hash passwords with a salt and then save them in our database.

For request authentication, passport.js provides us with an easy option to use JSON Web Tokens. These tokens contain the user's ID and are signed with a secret key. passport.js provides easy middleware to always be able to verify and access the user in any of our API endpoints by defining a custom function that extracts the user ID from the JWT and then fetches that user from the database.
This JWT gets sent to the user on registration/login and is saved in the mobile device's secure storage (Android's keystore or iOS's keychain). Whenever the frontend makes a request, this JWT is included in the request and allows our backend to verify the sender.

2.2.3 Media Processing

All processing of user uploaded media (i.e. images and videos) that is needed for our application is done server-side. We chose this approach because media processing, especially video transcoding, can be a very performance-heavy task and we wanted to make sure that our main app stays fast and smooth and doesn't cause unnecessary battery drain. We also generate multiple different qualities for uploaded content to ensure that users with bad internet connections can still view all of the content with as little waiting as possible.
When a user uploads a media file, it is first stored in a temporary location, and will be removed once processing is completed.

For image processing we use a popular Node.js module called `sharp` to convert images to jpeg to decrease file sizes and generate the different qualities in the form of multiple different resolutions.

To implement video streaming, we decided to go with the open-source HTTP Live Streaming (HLS) standard created by Apple. Unlike with more traditional video containers like mp4, in HLS videos are represented as a series of video segments of similar length and an index file that describes how the segments are to be put together by the media player. It also supports adaptive bitrate streaming by allowing segments to exist in multiple different qualities. A media player then just needs the main index file and can request the needed segments in any of the supported qualities (currently 720p, 480p and 360p).
To create these segments and index files, we use `ffmpeg` through the `fluent-ffmpeg` Node.js module. `ffmpeg` will create multiple HLS streams from an uploaded video, each one containing an AAC audio and an h264 video stream with different bitrates and resolutions.

Some of us already had some experience using ffmpeg which helped us a lot to perform this task.

2.2.4 Profanity Filtering

We are currently using profanity filtering for any text message that is send by any user.
The profanity filtering happens in our backend before the message gets saved into our database. The backend compares words in the message to the recommended list of forbidden words. If a certain word appears in that list, it gets replaced by `*****` and then saved to our database.

2.2.5 Automatic Testing

For automated testing we use Jest, an open-source JavaScript testing framework. Together with `supertest`, an adapter between the Express server and our unit tests, it's very straightforward to ensure the correctness of API endpoints. Both Jest and supertest provide infrastructure which in the end allows us to easily validate response codes and data, and ensure that things like authentication are working as expected.

These tests run on every commit in GitLab CI, together with the aforementioned formatter/linter, on a self-hosted GitLab Runner (since git.rwth-aachen.de does not provide CI runners on its own).

2.3 Frontend

2.3.1 Media Streaming

We use the Expo videoplayer component to handle the stream. Once supplied with the indexfile, it requests all further videosegments as required unassisted. To supply those index files we first fetch all metadata for all media files for a specific event. This metadata gets saved in our frontend but the video doesn't get fetched until it is actually being watched. This ensures that our application runs quickly and doesn't take a lot of time to load initially.
The videoplayer components handle all the buffering. Depending on the bandwidth and if the quality is set to auto, the component uses the index file to fetch a different video quality. If the user however requests a specific quality, only segments with that resolution are made available to the videoplayer, thereby locking it at the requested level. This allows the user to both not care about any quality settings if their internet connection is good enough and to also decrease it if they want to save transmission data while outside.

2.3.2 Design

The main objective of our design philosophy was effective communication of available features through UI/UX design, as well as preserving mobility. We wanted to anticipate the user's needs and make sure that they always have easy access to all relevant functionality at every step. Where possible we used recognizable symbols to make actions feel intuitive. We also tried to introduce coherent design patterns, where screens with similar functionality will have a similar look and feel. By this grouping we enable the user to transfer their already acquired skills instead of having to relearn a new interface.
For mobility we decided to introduce a tab bar as the main mode of navigation. This bar is always available and easy to access at the bottom of the screen. Should the user switch their interest to another part of the app, or want to look up some previous information from another screen, they always can. We hope this makes the app feel less restricting and more cooperative. Inbetween navigating tabs, the screen state inside of each tab is also preserved. This way the user can always pick up where they left of before leaving the tab, instead of having to navigate back manually.
All arising errors are communicated to the user via small notifications stating what went wrong. This way the user is always aware of the current state of the app and can take appropriate action. If, for example, an event picture could not be uploaded, due to a failing connection, the user will be made aware and can respond accordingly.

2.4 Live Streaming

One of the requirements for our application was the inclusion of a synchronized media playback feature. After a lot of brainstorming we came up with some ideas like a feature that would let you watch a synchronized slideshow of a past events media uploads with your friends or other attendees of the event. We felt however that all of these features were more gimmicky than actually useful features, so we decided to fulfill the requirement by implementing a video live stream (and chat) feature. This allows any user attending a running event to start a live video broadcast that can be viewed by anyone alongside the other media uploads for the event.
The app uses the `react-native-nodemediaclient` Node.js module to generate a Real-Time Messaging Protocol (RTMP) stream from the camera and display these streams in an events media section. On the server we use the `node-media-Server` module to accept and broadcast the user generated RTMP streams.

2.5 Content Recommendation

To recommend content, we introduce a ranking metric to impose a partial order on all events relative to a given user. After first filtering the list of events by the general availability, all remaining events are assigned a score based on how well they match the account in question in different categories. These categories are: the distance of the event to the user's current position, the number of tags of the events the user has identified as their favorite, the number of people the user follows at the event, the average rating of previous events from the host and the amount of media for the event.
These values are then weighted with configurable parameters to align their importance to p the users preferences. These parameters can be set in the "Profile" tab under "Manage Discover Feed". After computing the score for each feature, they are summed and then used to sort the events.

2.6 Push Notifications

To notify the user of certain important events, they can opt to receive push notifications. Currently, this includes notifications for new chat messages, new followers, and reminders when an event the user is interested in is starting. This can easily be expanded to other things/events.

We use Expo-provided packages/services as an abstraction layer on top of Google's FCM and Apple's APNs notification services. On startup the app retrieves a device-specific notification token, exchanges it for an Expo notification token, and sends it to the backend (provided the user is signed in); this token is stored in the database, associated with the user. The backend can then use those tokens to send notifications to specific users by interfacing with an Expo-provided service, which dispatches notifications through FCM/APNs accordingly, making cross-platform notification sending quite simple.

2.7 Privacy

We try to respect the users' privacy as much as possible by saving only the minimal amount of data we need. Any location data that is used by our application for geo-location purposes is not saved in our database and only used for calculations. We also make sure to not share any kind of personally identifiable information with other service providers. All of our data stays on our own server in our own database, and we do not collect any sort of analytics/metrics. Our API uses various checks to make sure that users can't access data that they are not supposed to access.

3 Frontend User Manual

3.1 Account Creation & Login

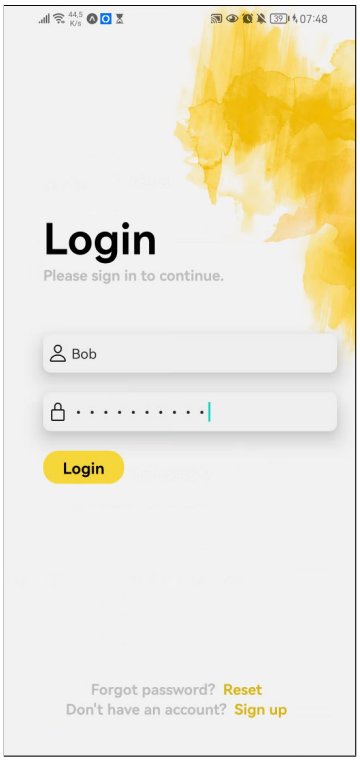

Figure 3: Login

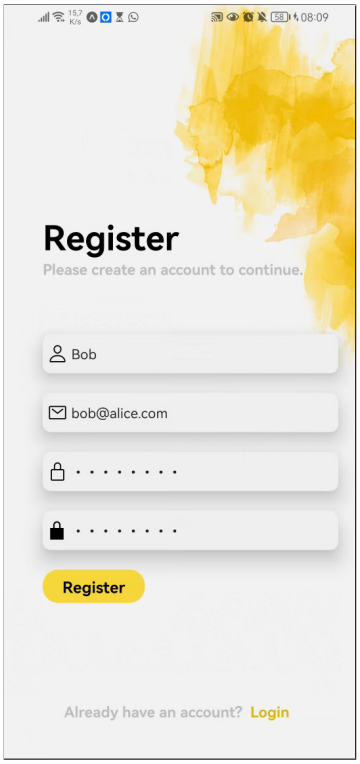

Figure 4: Register

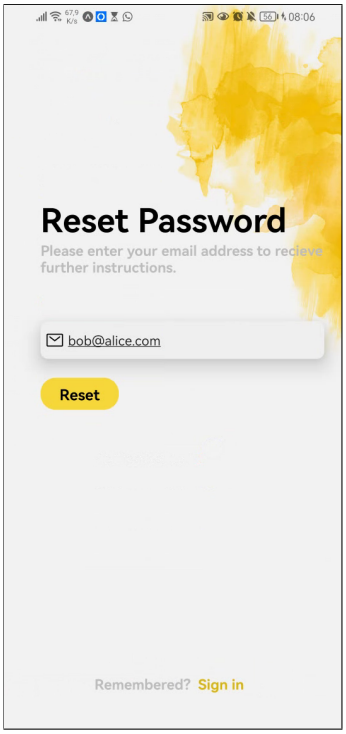

Figure 5: Forgot Password

As seen above we provide the usual functionalities around user accounts. The user has the option to always navigate back to a previous screen using a dedicated button at the bottom of the screen. We currently only provide usual username-password logins, however it would be easy to implement a Google login or other OAuth providers here as the authentication library we use (passport.js) supports this.
If the user forgot their password, they can simply type in their email and will get a new password sent to them via email.

3.2 Discover Events

Figure 6: Discover

Figure 7: Quality

Figure 8: Navigation

The Discover Screen is the **main way** for users to discover new events to visit. Our main goal here was to make sure that it feels as natural as possible for our users. That's why we designed this to function the same way like Instagram stories as there are a lot of people that use Instagram nowadays.

In 6 we can see the general structure of the discover screen. At the top the user sees how many media files are available for this event, in this case two. They also get an indication which media file they are currently watching. The type of media file can be seen in the top right corner, in 6 it's a video. It is also important to note that media files are ordered by their type. Livestreams are always first if they exist for an event, afterwards videos and lastly images. More information about the order of the events in the discover screen can be found in 'Content Recommendation'.

Furthermore, in 7 you can see that the user has the option to adapt the quality of the media files to their liking. You can find more information on how this works in the section about Media Processing and Media Streaming. They also have the option to mute videos and livestreams.

Let's talk about navigation. In 8 the different tapping areas are colored. The tapping zone on the left (red) lets you switch to the previous media file. The green tapping zone acts as a pause/play button. If you tap the right side (blue) you get to the next media file.
Switching to the next event also works like going to a new set of stories on Instagram by swiping to the right (or to the left for the previous event).
Clicking on the name and the profile icon on the upper left corner brings you to the event itself! You can find further information about this in 'Interacting with an event'.

3.3 Using the map

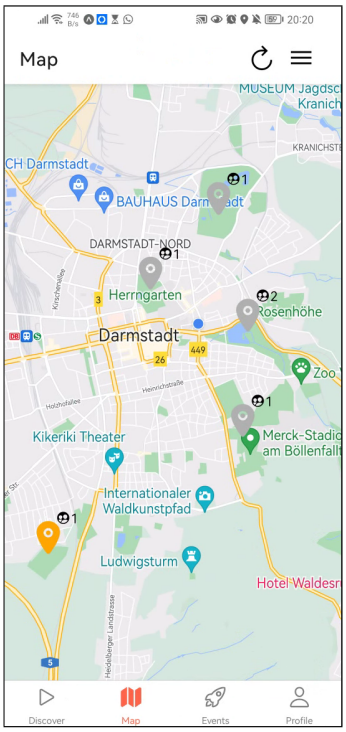

Figure 9: Overview

Figure 10: Preview

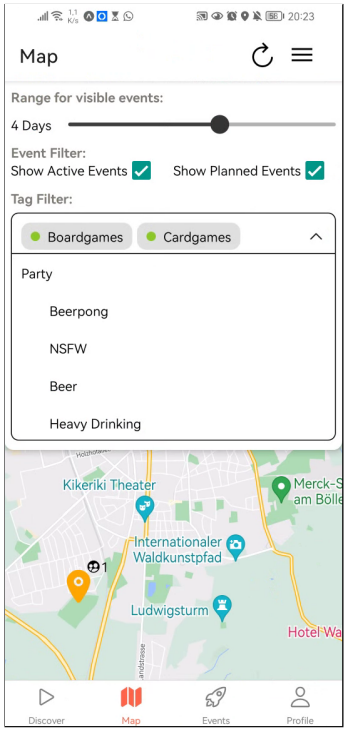

Figure 11: Filtering

The map screen is our second way to discover events. As you can see in Figure 9, there is a marker for every event in my close proximity. Active events are orange, events that have not been started yet are grey. There are also icons indicating how many people are attending the event or interested in the event. If there is a livestream I can also see it in this screen.
Figure 10 shows what happens when the user taps on an event. The user gets the most important information like when the event happens, how many people are going to be there and who the host is. They can open the chat, interact with the event and open the Event Details screen here.
The last screenshot shows some options for filtering events in the Map Screen. You can filter by a certain date range, only show active or planned events and also filter events by using tags. Events that don't match the filter criteria will not show as a marker on the map.

3.4 Interacting with an event

Figure 12: Media

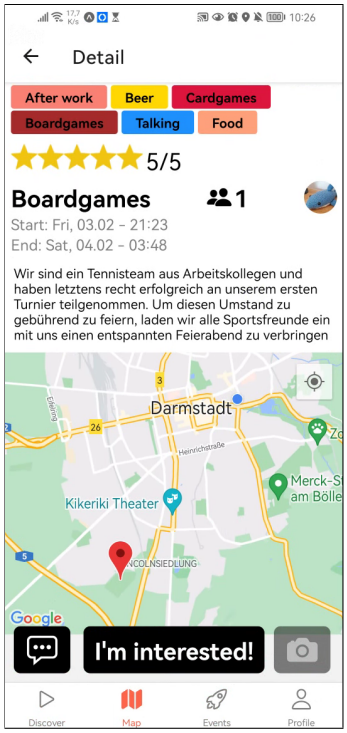

Figure 13: Event Info

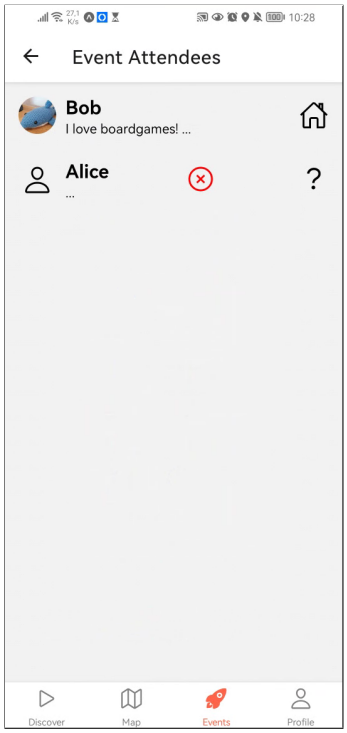

Figure 14: Attendees

The event detail view is the main way to interact with other people and events in our application.
In 12 you can also see another representation of media that is linked to this event. It works the same way like in the Discover Screen but is limited to media for this specific event.
The floating bar at the bottom of the screen features the most important buttons. One to access the chat (more about this in 'Interacting with other users'), one to add media if you are attending the event and the main button in the middle.
The main button in the middle has different functions depending on the state of the event. For example if you are not in range and/or the event has not started yet you can only claim that you are interested in the event. If you are in range (<50m) and the event has started, the button will say "I'm here" and you can use it to attend the event.
If the user scrolls down, they will able to see more information about the event (Figure 13. The following information can be accessed:

- Time of Event
- Average Rating of Event (if already active)
- Link to Host Profile and current Attendees/Interested people
- Description of event
- Map with event and own location to navigate to event
- Tags for this event

Everybody can access the list of attendees like seen in Figure 14. Depending on the icons next to the user's name, you can see if they are the host, attending, or just interested. The host can also use this screen to ban users, which will not allow them to interact with the event in any way (no chat function, no upload, etc.).

3.5 Add media to event

Figure 15: Upload

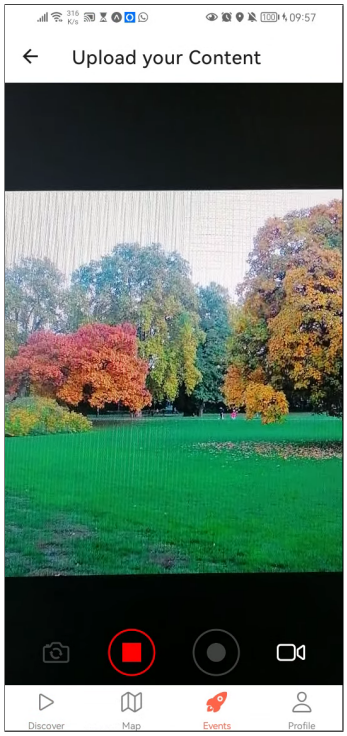

Figure 16: Video

Figure 17: Livestream

Media can only be added to an event by attendants or the host. We support different types of media upload. The different buttons in the screen support picture and video upload. The video camera icon on the right starts a livestream.
The user can also switch to their front camera by using the swap feature in the lower left corner.

3.6 Hosting an event

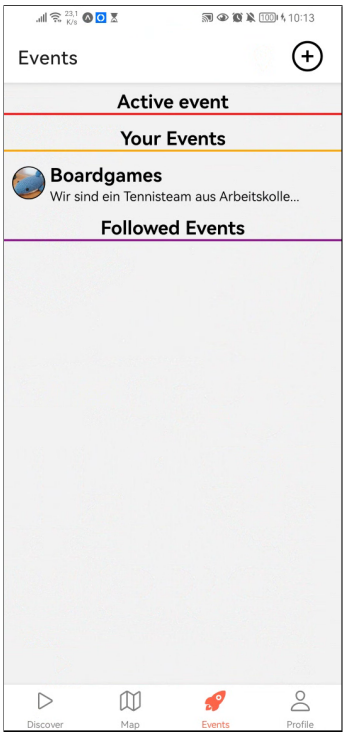

Figure 18: Overview

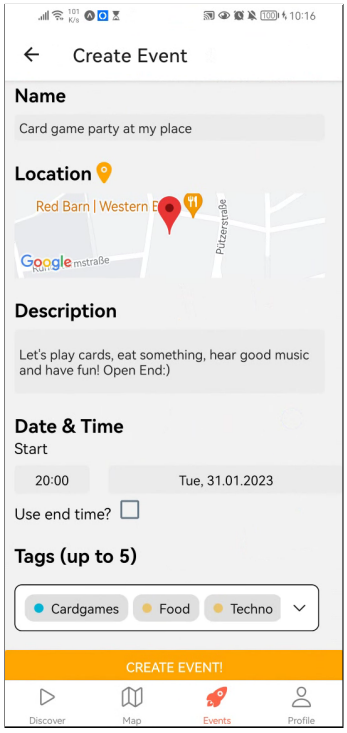

Figure 19: Inputs

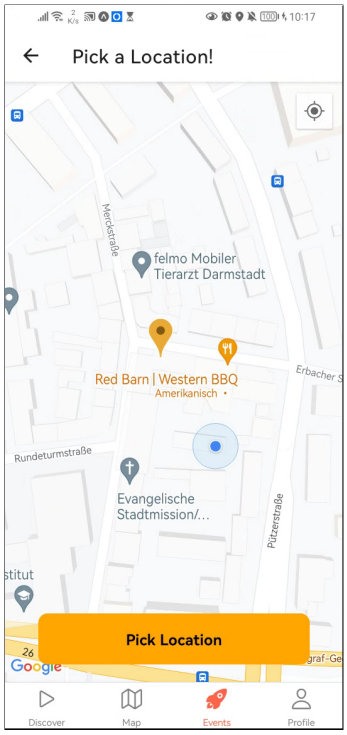

Figure 20: Location

To host an event, the user has to create a new event. This can be achieved by pressing the ”+” Button in the upper right corner of the Events Overview screen.
It opens up a screen with some inputs like the event name, event description, the specific time slot and a dropdown to select certain tags for your event. The user also has the option to create an open-end by not specifying any end-time. Our application uses native date and time pickers on all platforms.
You can also use the location picker to pick a location for your event by tapping the location icon.
If a host wants to edit an existing event, they can use the edit feature on the detail screen. This uses a similar layout as shown here, but the inputs are pre-filled from the database. This saved us a lot of code and makes it easier to maintain in the future.

3.7 Profile Screen & Interacting with other users

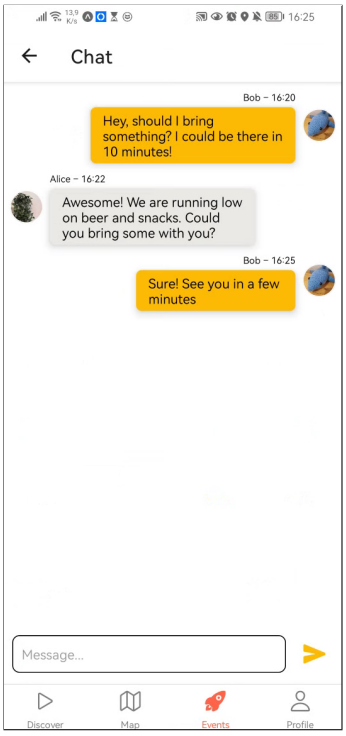

Figure 21: Chat

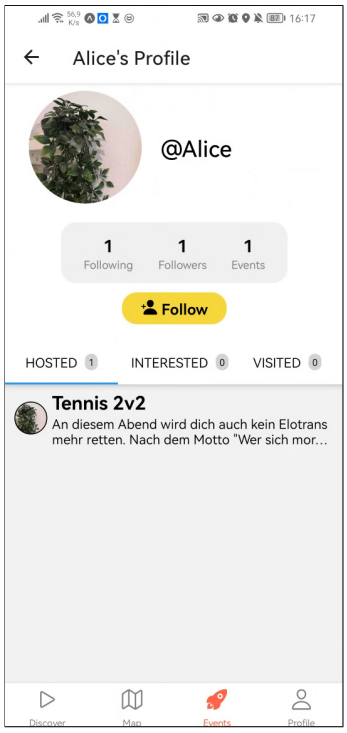

Figure 22: Profile

Figure 23: Followers

The main way of interacting with users, apart from meeting them at an event of course, are the event presentation with media and live stream functionality as well as the chat. While sharing media for an event or starting a live stream is a one-way route for communication, the chat feature allows other users not at the event to interact. If interested users then wish to, they can then get in touch with the people at the event using the chat feature. There you can ask more specific questions and see if your interest persists. This communication dynamic allows for quick filtering for the user. You should be able to tell at a glance if an event feels to your liking. This requires front-loading the interaction, as it is not reasonable to wait for a response for a first impression. But talking to the people there is quick and convenient if you want to do so.

The profile screen of other users allows you to view some information about them. You can see which events they hosted, in which events they where interested in and so on. A more passive/abstract way of user interaction is our follower system. Since the people you follow have an effect on what events are recommended to you, someone following you will have their discover feed altered to promote events you are interested in or attending.

However, the goal of evented is to connect and bring people together. It therefore does not encourage long conversations. Instead it wants you to meet new people and build new friendships face to face, our favorite kind of user interaction.

3.8 Profile/Settings Screen

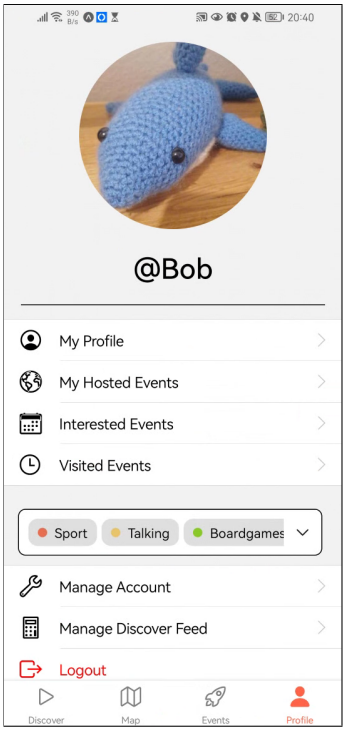

Figure 24: Settings

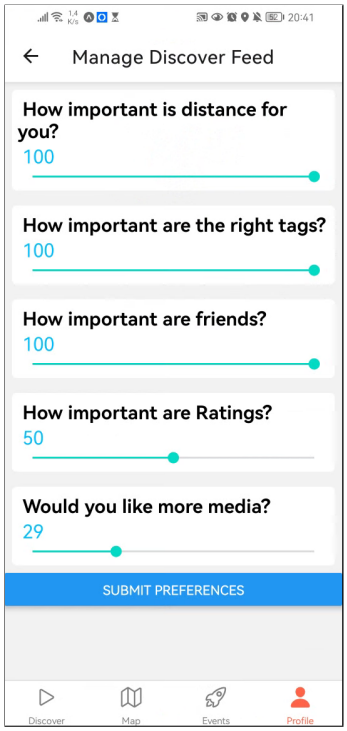

Figure 25: Recommendation

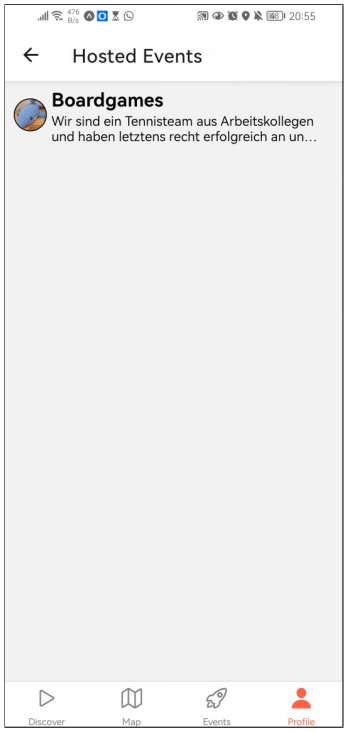

Figure 26: Events

The profile screen allows the user to change essentially everything that affects what they see (like recommendations) and what others see (profile picture, a short bio, username, etc.). Let's start with Figure 24. The user can view their own profile, as well as past and future events. They can also access other features, like the discover feed and account settings.

The recommendation settings as seen in Figure 25 are valuable input for our recommendation algorithm. The user can define how important certain stuff is for him. If they want to primarily see events where their friends are, they can increase the friends slider and decrease the others. Those values get used to sort the events in their discover screen.

This screen can also be used to access old events that you interacted with. If you want to look something up or chat with somebody at an old event, you can use this feature to access those events! Figure 26 shows what it looks like if you e.g. select 'My Hosted Events'.

3.9 Administrative Access

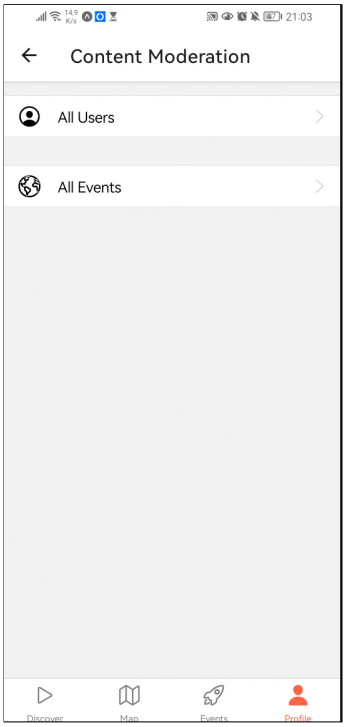

Figure 27: Administration

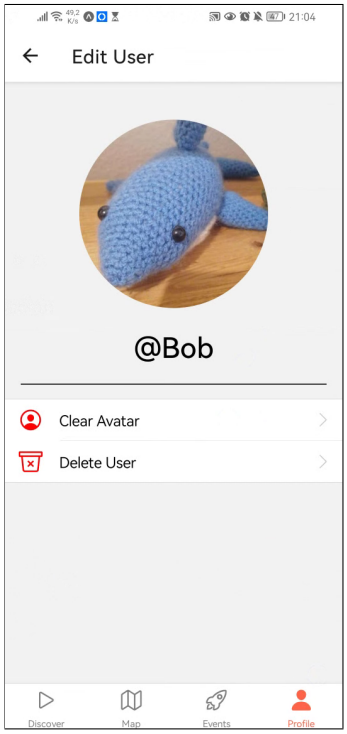

Figure 28: User

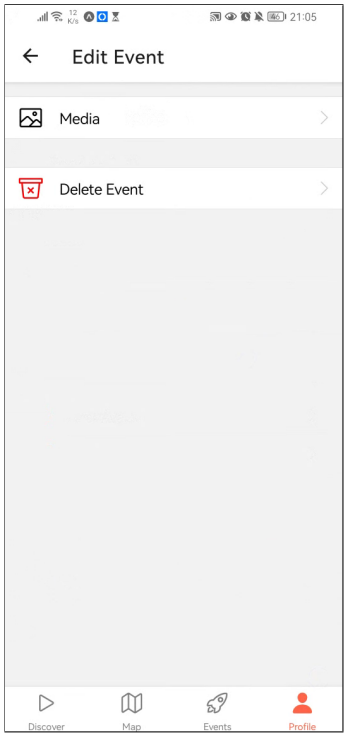

Figure 29: Events

If you are logged in as an administrator, you have access to the administration in the profile screen. This can be seen in figure 27. We allow the administrator to handle both users and events.
As seen in figure 28, the user first has to be selected from a list and can afterwards be updated. We gave the administrator the option to delete the users avatar and to completely delete the user account.
As seen in the last figure, after choosing an event from a list, we can either access all the media that has been uploaded to an event or delete the whole event. If we access the media we can select all the media from a list, take a look at it and if we find it inappropriate, we can delete it.

4 Conclusion

4.1 Problems

For the major part of our group, working on a mobile application was completely new, so we had to do a lot of research at the beginning regarding React Native and related technologies. Luckily, most of us already have some web experience, which helped during this process.

As synchronized playback didn't really fit the whole concept of our application, we talked the organizational team and asked if we could instead implement something different which would fit better. We fortunately came to an agreement and so we started researching how to implement live video streaming in our application. We ran into some issues with Expo Go though, which we fixed by going the more involved route of building a native application bundle ourselves (still using Expo). We were then able to get a functioning livestream in our APKs.

4.2 Outlook

Our application is not completely finished for sure and there are some aspects that can be improved. We also still have a lot of feature ideas that we sadly couldn't realized because of time constraints. The following lists contains some of both points:

- Add more languages to make it easier for users to use it in their primary language
- Add specific predefined events for games which then provide a game support application (like a money tracker for Skat, point tracker for Speedminton/Badminton, tracking for BeerPong, etc.)
- Using AI to also automatically tag or delete NSFW images/videos and improve profanity filtering in chat messages
- Improve the filtering on the Map Screen
- Include some type of monetization to ensure our application is sustainable for us
- Get application on the play store to get it to a large userbase and make it easy to install (Cross Platform already possible)
- Handle your own followers by being able to accept/remove them

4.3 Final conclusion

Working on something practical in a more 'work-like' way at university is always an enriching experience. We had the opportunity to try out new technologies like React Native and Expo which helped us a lot in further deepening our software development skills and getting first-hand experience in building mobile applications.

Using a iteration-based agile approach combined with the feedback from the 'customers' really helped a lot with the flexible requirements we had. Getting some feedback from outside always helps to see the whole application in a new light.

We are all really happy with the application we built in this short time and look forward to force some friends to use it to get to a critical mass (or maybe a lot of friends).