Tremola Extension: Meeting Coordination

A report
for the project "Tremola Meeting Coordination"
accompanying the lecture "Internet and Security FS22"

Gianni Rajsic, André Vieira Ribeiro, Mark Starzynski

Universität Basel
17.07.2022

Sections and Chapters

Contents

1 Introduction

For the "Internet and Security" lecture's accompanying project we have decided to implement a meeting coordination extension for the Tremola app.

We have chosen this project because we wanted to do something with Tremola and are interested in the Secure Scuttlebutt protocol. Furthermore the idea of developing something that will be run on a phone was also a motivation.

Although we would have liked to implement a chess game, we have noticed that a lot of games were already chosen by other groups, so we thought it would be better to implement something different to add more value to Tremola. We have settled to try and add scheduling coordination functionality to the app because it was a suggestion already and also introduces additional complexity (more than two participants) to the project, which will ultimately enhance our learning experience.

The following chapters will guide you through our process from research to execution up to the lessons learned and our final result.

2 Research and Planning

To implement our extension we had to do some research first. Very early on we started to learn about Kotlin (see references [**4**] and [**5**]), which even though we all were unfamiliar with we could relate well to because its based on Java (which we have a good understanding of already). We also tried to familiarize ourselves with the Tremola code and started thinking about how our extension should be implemented into the GUI. Since Tremola is built on top of the Secure Scuttlebutt protocol we also looked into the documentation [**1**]. Further we looked into the web development trifecta HTML, CSS and JavaScript ([**2**], [**3**] and [**6**]), which proved to be quite relevant to our project.

We have had some issues with getting the development environment running for all team members. Furthermore we needed to find solutions to enable all members to be able to properly run the app on two mobile devices. This was crucial for us, since most of the time we planned out for the project would be spent online and we still wanted to ensure that all members could test and debug their progress.

Our planning for the project was to research it and think about good implementations as early as possible (during the semester) and really do most of the coding for the project after the exams. This semester was special in the sense that we had two projects to complete (the other being for the OS course). The very generous late deadline gave us almost three weeks after the semester which we could fully dedicate to our extension. With the expected 40 hours of work per member this time frame had a lot of leeway to even exceed this requirement should it be necessary.

3 Execution

The execution of the project was divided amongst the project group members in the following:

	G. Raijsic	A.V. Ribeiro	M. Starzynski	All
Research				XX
Prototyping				X
Frontend	XX	XX	X	
Backend			XXX	
Testing/Debugging				X
Code Documentation				X
Report			X	
Manual		X		
Video	X			

4 Implementation

This chapter is designed to give the reader an outline about how the various steps were done and what we had to do to complete our project and get a working extension running.

4.1 Development Environment and Prototyping

Setting up the development environment was pretty straight forward, although it came for some group members with specific issues like for example the Android Debug Bridge daemon malfunctioning for some reason, which didn't allow for proper testing with a device. All those issues could be solved with time and the internet though.

In this phase we have mostly studied the Tremola code and tested the app on actual hardware. First prototyping included sending specific code messages via implemented `priv:post` keyword and trying to send encoded information to specific recipients. For further explanation see the upcoming chapter about the backend.

Then later on, with some basic GUI elements added, we have started to process the information sent directly without displaying any messages up to the point, where we were able to run one meeting poll per group. This was later on expanded to include unlimited meeting polls.

4.2 Secure Scuttlebutt Protocol

Very early on we've read through the complete guide about the Scuttlebutt protocol [1] to be prepared for the upcoming work on Tremola. Although not as necessary as we thought it would be, since the focus of all Tremola projects would be more on the frontend side (as Professor Tschudin also pointed out in the Tremola introduction meeting), we still benefited from it when reading the Tremola code. For example it was easier to follow the path from user input in the GUI up to the creation and distribution of the according log entries when we already knew about Scuttlebutts protocol formats, especially when studying the backend written in Kotlin.

4.3 Tremola Extension "Backend"

As already mentioned the project is mostly focused on the frontend, considering the overall architecture of the Tremola app. For the purpose of our project we will refer to the backend as the processing and storing of information given by the user via GUI. In addition some minor changes to the actual backend in Kotlin were made to enable us to separate our entries from normal posts.

More specifically, we've created 2 different content types `meet` and `vote` before the signing process. We considered first to just encode our data into a post string, but decided it was cleaner to change the type. In retrospect, we could have probably used just one content type instead of two and encode various use cases like creating a new meeting or submitting a vote within this one type only.

4.3.1 Processing. For the processing of a user input we looked at how Tremola did it and implemented it in a similar manner. For various interactions by the user with the GUI there are functions in `tremola_ui.js` which can fetch the inputted data if needed and send it with a command keyword in a specific format to the backend. For example creating a new meeting calls `create_new_meeting()`, fetches the title and considered dates, Base64 encodes and sends it as "`meet: dates btoa(data) recps`" to the backend (btoa() -> encoding).

This string then gets rerouted to the WebAppInterface where the content gets formatted in the right way, chained to the last entry and a signature is added (see `SSBmsgTypes.kt`). The log entry then is added locally (persist), sent to the frontend for possible actions and streamed to all peers, where the same process occurs for all relevant recipients (see `WebAppInterface.kt:rx_event`).

4.3.2 Storage. After a log entry gets reformatted as JSON object and sent to the frontend (see `WebAppInterface:sendEventToFrontend`) via `eval()`, `b2f_new_event(event)` handles the browser-side storage and any further frontend actions based on the content type. For our feature we have extended the existing Tremola storage with an entry `"polls"` as seen in `tremola.js` in `resetTremola()`. For the structure of the poll database see 7.1 Poll Database Format in the appendix of this document.

The polls are stored after group chat (-> `conv_name`), each meeting poll has its own `meetID` where all data related to it is accessible once made persistent. Relevant data per poll stored includes the dates within a 10x3 array, how many votes a date has already (stored in a "votes" array), "voteCounter" for how many votes were made, the creator of the meeting poll, members of the group chat and hence participants in the poll, each separate vote within "membervotes" and when the poll was created locally.

All this information then gets stored via `persist()` permanently so we can access the data even when the app is restarted.

4.4 Sorting

For our feature of listing all meeting polls we wanted to be able to sort them after most recent. We first decided to implement it in the same way group chat messages are implemented and sort via locally generated timestamp. This is the current implementation, although we have tried to implement it additionally with Scuttlesort because of its advantages, we couldn't get it done in time and had to prioritise other parts of the project. On this note we would like to thank Etienne Mettaz for all the help he provided us with in the Zoom meetings.

4.5 Frontend and Extension Features

After succesful data storage we had to ensure this data gets used properly with every user interaction. The user is able to create meeting polls for every group they are in. The meeting poll is specific to the group, meaning the participants of the poll are the members of the group chat. Upon creation of a new meeting poll each member can access it via `Show meetings` button in the overlay menu. Displayed will be a list of meeting polls specific to the group, sorted after most recent. Since Scuttlebutt is append-only, users can - synonymous to the way group chats are implemented - "delete" a meeting poll by "forgetting" it. A poll is completed if every member of the group has voted or if the meeting poll is finished by the creator. For further details please see the separate extension manual PDF.

Most noteworthy to us was the manipulation of HTML content via element property `innerHTML`, which allows us for example to change buttons within JavaScript files.

5 Lessons Learned

We have learned about Kotlin and the Secure Scuttlebutt protocol and although this proved to be less relevant for the project overall it was still worthwhile research, especially the very pretty and informative Scuttlebutt guide in regards to the lecture.

We have programmed predominantly with Javascript, HTML and CSS with a little Kotlin on the side. We took our experience with IntelliJ IDEA from previous projects and further expanded upon it with Android Studio. We got a glimpse into mobile app development and experienced the joy of running something that we made on a mobile phone.

We were able to apply the processes of proper software design, which we learned from previous courses, on a practical level. From understanding Tremola and acquiring the fundamental knowledge needed, creating various mockups, prototyping, reiterating on ideas, version control and testing up to the final product and its documentation.

6 Conclusions and Future Upgrades

We have reached our goal of creating a functioning meeting coordination extension and are quite happy with the current end result. This feature is, in our eyes at least, by no means perfect and we had a lot of ideas to further improve upon it. Some of which are:

- A prettier user interface, especially the create meeting form
- Additional functionality for polls
- Implementation of Scuttlesort
- Database optimization
- Notifications and badges
- Remove limitation of fixed amount of dates and make it dynamic
- Replace counter with names and individual votes. This is already implemented in the local storage.

Given we happily exceeded the expected time requirements for this course project by more than just a few hours and the deadline being right around the corner, all good things have to come to an end. We hope this extension might be of some use to Tremola.

7 Appendix

The documented code for the full project can be found in the separate zip-File uploaded to the dedicated place. In the archive as well you will also find a manual and a short video which guides you through the extension and its features. The video can be seen online here: [YouTube LINK]

Furthermore a pull request was made in addition for an easier overview of the changes relative to the original Tremola project.

7.1 Poll Database Format

```
tremola.polls[conv_name] = {
"12512979": { // meetID entry
        "meetID":"12512979",
        "alias": conv_name, // concatenated string of group member IDs
        "title":"The title of the meeting poll",
        "dates":[
                [<title, creator, meetID],
                [date1,startTime1,endTime1],
                [date2,startTime2,endTime2],
                [date3,startTime3,endTime3],
                [date4,startTime4,endTime4],
                [date5,startTime5,endTime5],
                [date6,startTime6,endTime6],
                [date7,startTime7,endTime7],
                [date8,startTime8,endTime8],
                [date9,startTime9,endTime9],
                [date10,startTime10,endTime10]],
        "votes":[0,0,0,0,0,0,0,0,0,0],
        "voteCounter":0,
        "forgotten":false,
        "finished":false,
        "creator":"@oFdW5FtddZ2EDG/p7k2PQqMyFDeBFsLvafrfBUSRazQ=",
        "members":[ // e.confid.recps
                "@fAfgYutLe7jw7wS+AX35QBKKByFYydsY5JTQQi//C/E=.ed25519",
                "@oFdW5FtddZ2EDG/p7k2PQqMyFDeBFsLvafrfBUSRazQ=.ed25519"
                ],
        "membervotes":{
             // ID of voter 1
                "@fAfgYutLe7jw7wS+AX35QBKKByFYydsY5JTQQi//C/E=":
                    ["touched", 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
            // ID of voter 2
                "@oFdW5FtddZ2EDG/p7k2PQqMyFDeBFsLvafrfBUSRazQ=":
                    [999] // 999: not voted yet
                },
        "touched":1658317705462
        }
}
```

References

[1] Scuttlebutt protocol guide. URL `https://ssbc.github.io/scuttlebutt-protocol-guide/`.

[2] Javascript course. URL `https://www.sololearn.com/learning/1024`.

[3] Html course. URL `https://www.sololearn.com/learning/1014`.

[4] Kotlin course. URL `https://developer.android.com/courses/android-basics-kotlin/course`.

[5] Michael Kofler. *Kotlin*. Rheinwerk Computing, 2020.

[6] J.N. Robbins. *HTML5 Pocket Reference*. O'Reilly, 2013.